MASTER EN SCIENCES INFORMATIQUES

Les standards de communication dans l'environnement industriel

Peters, Marc

Award date:
1992

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Les standards de communication dans l'environnement industriel

par

Marc Peters

Licence et Maîtrise en Informatique

Mémoire de fin d'études

Année académique 1991-1992

Promoteur : M. le Professeur Ph. van Bastelaer

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Je tiens à remercier tous ceux, qui, par leur aide et leurs remarques constructives, m'ont permis de réaliser le présent travail de fin d'études.
En premier lieu je remercie mon promoteur, Monsieur le Professeur Ph. van Bastelaer qui a accepté de me promouvoir; je le remercie pour le temps qu'il m'a consacré.
Mais remerciements vont aussi à Monsieur le Dr. K. Weber, Monsieur W. Werner, Monsieur H. Netta et globalement à la firme Siemens Amberg. Pendant les cinq mois de stage, j'ai eu la chance d'apprendre beaucoup.
Je remercie également Madame C. Bruel ainsi que ma soeur pour leur travail de lecture et de corrections de forme et d'orthographe.
N'oublions pas non plus ma famille, mes amis et tous les autres non mentionnés directement pour leur soutient, leurs conseils et leur effort de compréhension.

"The aim of Open Systems Interconnection is to allow, with a minimum of technical agreement outside the international standards, the interconnection of information processing systems

from different manufacturers under different managements of different levels of complexity of different ages."

(ISO/DIS 9506-1 2/11/1988 page 7)

Table des matières

Préface

Depuis quelques années, l'informatique pénètre dans le processus de fabrication de nombreux produits. En raison de la baisse des coûts du matériel informatique et de la montée de nouveaux concepts tels que les réseaux locaux, on ne se contente plus d'automatiser le processus de fabrication. On voit de plus en plus l'intérêt d'interconnecter les différents automates par un réseau. Ceci mène ainsi à une fabrication intégralement contrôlée par l'ordinateur.

Lorsqu'une telle évolution a lieu, il est normal de voir apparaître une multitude de systèmes incompatibles entre eux. Mais comme on désire être indépendant de tout fabricant et qu'il est parfois nécessaire ou même souhaitable de pouvoir interconnecter des équipements de divers fabricants, les efforts vont se concentrer tôt ou tard sur le développement d'un standard.

Ainsi, le présent travail introduit le lecteur de manière simple et intuitive dans la problématique (L'introduction). Après avoir expliqué le principe, nous abordons, dans un deuxième chapitre (Aperçu des normes internationales), les standards internationaux pour la communication dans le domaine de l'automatisation. Nous y expliquons, entre autres les concepts MAP et MMS. Ceci fait, nous descendons un peu l'échelle pour présenter au cours du troisième chapitre (Une norme nationale allemande : Le PROFIBUS) une norme nationale allemande qui est le PROFIBUS, avant d'aborder brièvement au chapitre quatre (L'architecture utilisée par la SA SIEMENS) les normes développées par SIEMENS notamment pour la communication entre leurs automates programmables.

Le cinquième chapitre donne ensuite un aperçu du travail durant le stage effectué chez SIEMENS et explique une démarche possible de conception et de développement d'une couche de transport (comparable à la couche 4 du modèle OSI).

Comme il s'agit d'une matière assez complexe et peu abordée dans les publications, nous avons essayé de nous référer le plus possible au modèle OSI qui est le standard pour l'interconnexion de systèmes ouverts.

L'illustration au moyen de nombreuses figures permettra de mieux comprendre les idées parfois difficilement exprimables en langue naturelle.

Dans la thématique abordée, il est courant d'employer beaucoup d'abréviations. Pour cette raison et pour ne pas devoir mettre après chaque abréviation sa signification, nous donnons au lecteur au chapitre 7 une liste des abréviations utilisées tout au long du travail. Cette liste reprend les abréviations, leur signification et si nécessaire et possible une traduction en français.

Néanmoins, lorsqu'une abréviation est employée pour la première fois, nous avons essayé de donner le nom complet. Lorsque cela n'a pas été fait, une astérisque indique que la définition se trouve dans la liste des abréviations.

De plus, pour ceux qui ne sont pas aussi familiarisés avec la télématique (ou simplement pour mémoire) se trouve au chapitre 8 une liste qui donne la signification de certains termes non expliqués dans le travail lui-même.

A ceci vient s'ajouter, après la bibliographie, un index qui permet de retrouver certains passages de manière plus aisée .

Ce travail peut donc permettre au lecteur débutant dans la matière de s'y introduire. Celui qui connaît bien la thématique pourra utiliser le travail pour approfondir ses connaissances ou pour découvrir l'existence d'autres standards que les standards internationaux.

Si l'on désire une information plus détaillée sur certains aspects (notamment pour les protocoles utilisés, le modèle OSI, les réseaux locaux, etc.), on trouvera de nombreuses références dans la bibliographie .

1. Introduction[1]

Lorsqu'on parle de communication au sens informatique, on pense souvent à l'échange d'informations entre ordinateurs. Ces échanges d'informations par l'intermédiaire d'ordinateurs existent aujourd'hui au sein de toutes les grandes entreprises et même entre différentes entreprises. De nos jours, tout le monde a déjà ne serait-ce qu'entendu parler de termes tels que courrier électronique (*electronic mail*) ou EDI* (*electronic data interchange* - échange de données informatisées) .

A côté de cette communication de type dit bureautique s'est installé et s'installe encore actuellement un autre type de communication dans les entreprises. Il s'agit de la communication entre différents appareils de production du même constructeur ou de constructeurs différents dans les grandes usines.

En fait, après l'automatisation partielle ou totale du processus de fabrication des produits, on voit de plus en plus apparaître la nécessité et/ou l'intérêt d'une communication entre les différents appareils d'automation. Si on pousse encore plus loin cette idée d'un échange d'informations entre ces appareils, on aboutit à un système GPAO* (Gestion de Production Assistée par Ordinateur), mieux connu sous le sigle anglais CIM* (*Computer Integrated Manufacturing*). Dans ce genre de système, presque tout le processus de fabrication est exécuté et contrôlé par l'ordinateur. En d'autres mots, à partir de la conception des pièces de fabrication faites sur un système CAD* (*Computer Aided Design*) toute une démarche entièrement informatisée est mise en route, démarche dont le résultat est la fabrication automatique des pièces.

Le but du présent travail est de présenter et d'expliquer différents standards de communication qui existent pour réaliser un tel système CIM. Partant du standard international MAP* (*Manufacturing Automation Protocol*), qui constitue une espèce de référence en la matière, nous terminerons par la description d'une démarche de conception d'une partie du module de communication nécessaire à permettre la réalisation du CIM. En cours de route, nous essaierons de révéler les différences entre les différents standards internationaux, nationaux et privés. De plus, nous tenterons de justifier la coexistence des standards en justifiant leurs différences.

1 Nous nous sommes basés principalement sur les références /1/, /10/ et /13/.

Avant d'entamer plus en détail les différents standards, nous nous attarderons un peu sur la notion de CIM afin d'expliquer son principe pour que l'on ait une idée de quoi il s'agit.

Pour ce faire, imaginons une usine qui fabrique un produit. Ce produit est composé de plusieurs pièces ensuite rassemblées.

Tout le processus de fabrication du produit, de sa conception à la production proprement dite, peut être divisé en différents étages. Au niveau le plus bas, nommé dans la suite niveau 1, on accomplit seulement des actions élémentaires de production (mesurer, actionner). Ces actions de base sont contrôlées par des contrôleurs de processus spécifiques. Le niveau 2 est formé par ces contrôleurs.

Comme le produit est une composition de différentes pièces, les pièces sont produites indépendamment l'une de l'autre. Par conséquent, on trouvera différentes cellules de production dans l'usine. Dans chaque cellule, un contrôleur de cellule contrôle la fabrication d'une pièce autonome. Ces contrôleurs de cellule se situent à un niveau, appelé niveau 3.

Si on continue à suivre le processus, on se trouve déjà au niveau de l'usine. A ce niveau, niveau 4, on contrôle les différentes cellules, mais on a aussi d'autres activités telles la conception (CAD), le CAQ*, le CAM* et la gestion automatique du stock.

Un niveau 5, niveau d'administration, va généralement compléter le système CIM, mais n'est pas pris en considération ici.

Il est évident qu'entre les différents niveaux il y a un trafic dense d'informations de tout genre. (*fig. 1.1. Hiérarchie de communication de la production automatisée*)

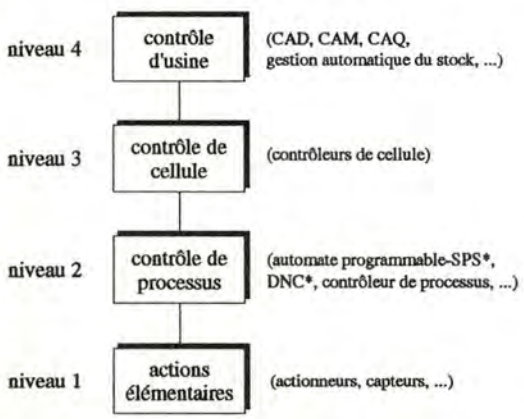


fig. 1.1. Hiérarchie de communication de la production automatisée

Pour aider à mieux percevoir le système CIM, la figure 1.2. reprend ce qui a été dit et illustré par la figure 1.1. en donnant une structure imaginaire des différents niveaux d'une usine et en y mettant des appareils types pour chaque niveau.

En outre, nous nous sommes efforcés de donner un nom significatif à chaque niveau. Ces noms peuvent être différents de ceux que l'on trouve parfois dans la littérature. En effet il fallait bien faire un choix. D'une part, les différents auteurs ne sont pas d'accord dans leurs dénominations des niveaux et, d'autre part, la quasi absence de littérature francophone sur ce sujet accentue cette difficulté.

Au niveau 1, désormais nommé niveau de terrain ou niveau de processus, on retrouve des capteurs, des actionneurs, des robots, des CNC*, etc.. Le niveau 2, nommé niveau de station, inclut des contrôleurs de processus tels que les automates programmables (SPS*), les commandes robots, les DNC*, etc.. Les contrôleurs de production et de cellule se trouvent au niveau 3, appelé niveau de cellule. Le niveau 4, enfin, est le niveau d'usine et on y trouve le ou les contrôleurs d'usine, les stations CAD, les BD*(Bases de Données), etc.. (*fig. 1.2. Structuration possible des niveaux*)

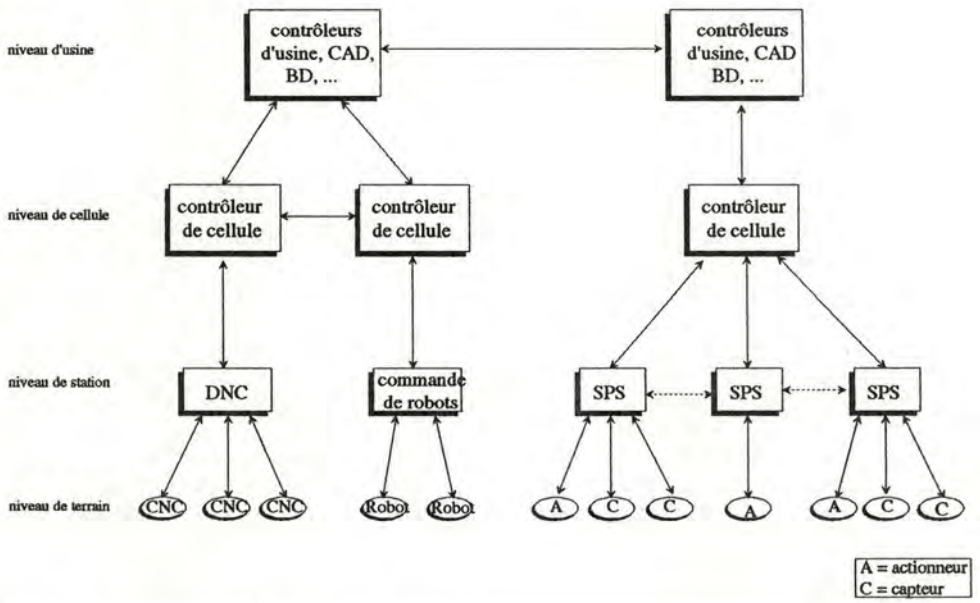


fig. 1.2. Structuration possible des niveaux

Jusqu'à présent, nous n'avons pas encore parlé de la structure physique des niveaux. Il est évident qu'afin de pouvoir échanger des informations entre les

différents niveaux, des connexions appropriées doivent exister. La figure 1.3. présente donc une architecture physique possible qui se base directement sur le concept de réseaux locaux. Ces réseaux locaux sont en fait les moyens de communications, d'une part, entre les appareils d'un même niveau et, d'autre part, entre les appareils de différents niveaux.

On y trouve notamment le réseau de terrain (*Field Bus*) qui relie les éléments du niveau 1 entre eux et avec le niveau 2, le réseau d'atelier (*Cell Bus*) qui relie les éléments du niveau 2 avec le niveau 3 et enfin le réseau d'usine qui joue le rôle d'épine dorsale (*Backbone Network*) auquel sont connectés le niveau 3 et le niveau 4. (*fig. 1.3. Structure physique possible*) Ce sera aussi par ce réseau d'usine qu'on pourra imaginer un accès à d'autres réseaux (des MAN* et WAN*) pour entrer en communication avec le reste du monde. Le passage d'un réseau à un autre est réalisé dans cet exemple par des passerelles (*gateways*) spécialisées. En réalité, cette fonction de passerelle est souvent assurée par des contrôleurs de cellule.

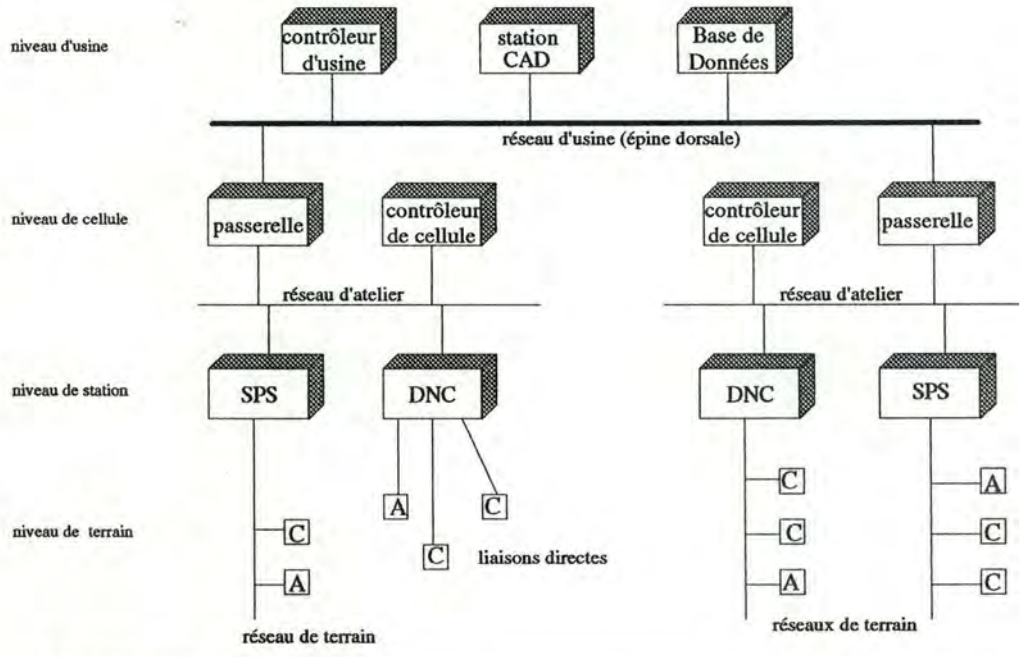


fig. 1.3. Structure physique possible

Enfin, la figure 1.4. nous montre les différents standards de communications utilisés aux différents niveaux de manière à permettre un échange approprié de données. On y retrouve notamment un réseau *MAP-broadband** comme épine

dorsale, un réseau *MAP-carrierband** prévu pour le réseau d'atelier et un réseau PROFIBUS pour le réseau de terrain.

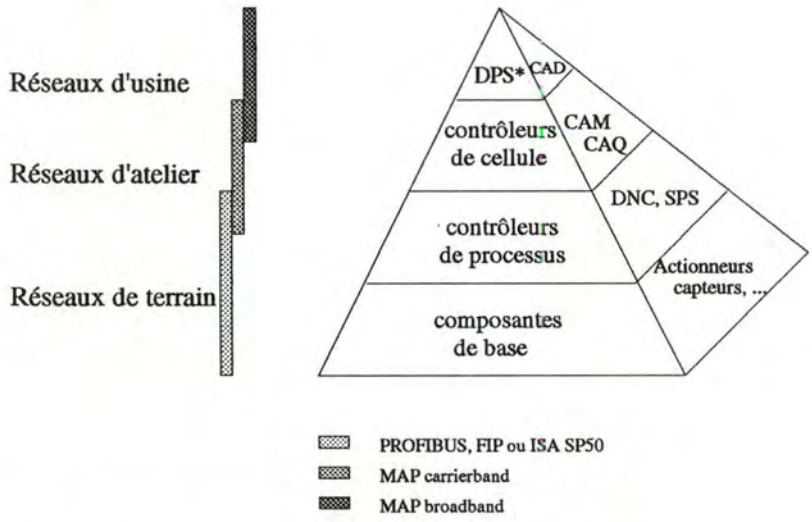


fig. 1.4. Les standards dans la communication hiérarchique.

Après cette introduction et cette explication de la notion d'un système CIM, les chapitres suivants vont présenter plus en détail les standards qui existent et qui permettent de réaliser ce genre de communication.

2. Aperçu des normes internationales[2]

Après la présentation de la problématique et du but de ce travail de fin d'études, le chapitre 2 donne un aperçu de ce qui existe comme normes internationales en la matière.

Après un bref rappel du modèle OSI pour l'interconnexion de systèmes ouverts, nous présenterons ensuite l'architecture et le principe de MAP/MMS, de MAP/EPA et de Mini-MAP avant de terminer ce chapitre par une brève présentation de TOP.

2.1. Rappel du ISORM et des principes de ce modèle

Le modèle OSI étant bien connu, et l'objectif du travail n'étant pas d'expliquer ce modèle en détail, nous nous contenterons ici de rappeler brièvement la structure et le fonctionnement de ce modèle qui constitue une référence en matière de communication entre systèmes ouverts. Pour plus de détails, le lecteur est invité à se reporter notamment au texte officiel de cette norme internationale (ISO 7498) ou aux nombreux travaux traitant du sujet.

Le modèle en 7 couches

Le ISORM (*ISO Reference Model*) est un modèle en 7 couches (*fig. 2.1. Les 7 couches du modèle OSI*) dont le but est de définir un standard pour l'échange de données entre deux ou plusieurs stations hétérogènes via un réseau.

La séparation en 7 couches est tout à fait arbitraire et on pourrait imaginer d'autres divisions (ce qui est d'ailleurs le cas pour TCP*). Le but de cette séparation en couches est de simplifier la transmission d'informations d'un système à l'autre. Pour chaque couche, l'ensemble des fonctions et comportements est défini, mais le fonctionnement interne de chaque couche n'est précisé nulle part. Le tableau 2.1. reprend les 7 couches et précise les fonctions de base.

2 Nous nous sommes basés principalement sur les références /9/, /10/, /13/, /16/, /18/ et /22/.

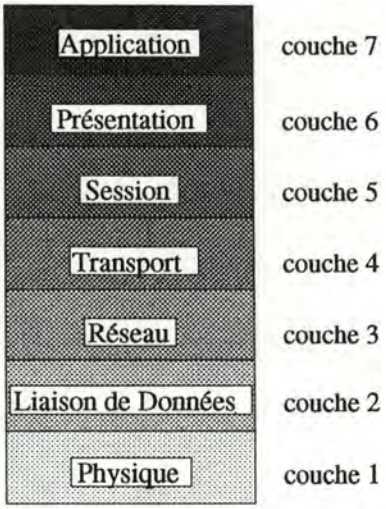


fig. 2.1. Les 7 couches du modèle OSI

couche 7	Les fonctions essentielles de la **couche application** sont de donner accès à l'environnement OSI et de prendre en charge toutes les fonctions qui ne sont pas réalisées par les couches inférieures.
couche 6	La **couche présentation** prend en charge les problèmes liés à la représentation des informations.
couche 5	La fonction principale de la **couche session** est de d'organiser la coopération harmonieuse entre les différents systèmes. Elle réalise les fonctions nécessaires au support du dialogue entre processus.
couche 4	Les fonctions essentielles de la **couche transport** sont d'effectuer un contrôle et une optimisation du transport des données de bout en bout. La couche transport est la première couche qui opère de bout en bout.
couche 3	La fonction de base de la **couche réseau** est de s'occuper de la transmission optimale des données dans le réseau, de s'occuper du routage et de procéder, le cas échéant, à un contrôle de flux et à un contrôle d'erreurs.
couche 2	La fonction de base de la **couche liaison de données** est de gérer la transmission des informations sur la ligne physique et de réaliser un contrôle de flux et un contrôle d'erreurs.
couche 1	La fonction principale de la **couche physique** est de s'occuper de la transmission physique des informations et de jouer ainsi le rôle d'interface entre les stations et le support physique

tableau 2.1. Les fonctions de base des 7 couches du modèle OSI

Les principes sous-jacents

L'objectif de chaque couche du modèle OSI est d'offrir des services à la couche supérieure. Pour fournir ces services, la couche en question utilise les services des couches inférieures. Ces services sont accessibles à des points d'accès aux services (*Service Access Points* - SAP).

En général , on peut dire qu'une entité N d'une machine A communique avec une entité N d'une machine B (l'ensemble des entités N forme la couche N). Les règles à respecter par les deux partenaires sont définies au bit près par le protocole du niveau N.

Ce n'est qu'à la couche 1 (couche physique) que les données sont réellement transmises.

Les informations échangées entre deux entités du même niveau sont appelées *Protocol Data Unit* (PDU*). *(fig. 2.2. Schéma de principe général)*

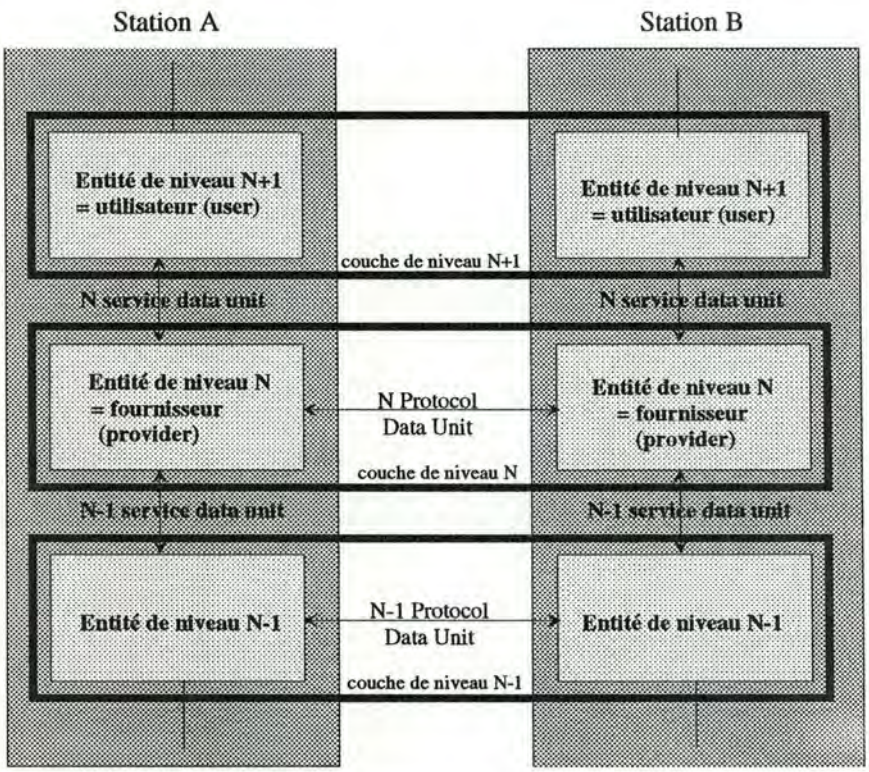


fig. 2.2. Schéma de principe général

Echange d'informations entre des processus d'application

La situation suivante se présente pour l'échange d'informations entre deux processus d'application. Une information est donnée par l'application à la couche application. Celle-ci ajoute aux données nettes qu'elle reçoit, des informations nécessaires en respectant un protocole tel que l'entité paire sache de quoi il s'agit et de quelle manière elle doit réagir. L'information ainsi étendue est passée comme information nette à transmettre à la couche inférieure (couche présentation) et le

processus se répète jusqu'à ce que l'information de départ avec toutes les extensions ait été transmise physiquement à la station paire. Dans cette dernière station, le processus se déroule à l'envers et chaque entité enlève l'information qui lui est désignée avant de passer le reste de l'information à la couche supérieure. A la fin du processus, l'information est donnée à l'utilisateur de la couche d'application sur le système pair. (*fig. 2.3. Echange d'informations entre processus d'application*)

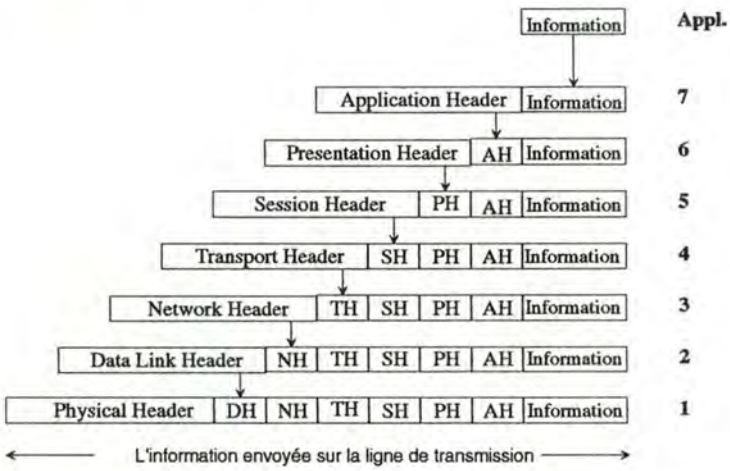


fig. 2.3. Echange d'informations entre processus d'application

Les informations ne doivent pas seulement être échangées directement entre deux stations terminales. Il existe souvent la nécessité de passer par des stations intermédiaires avant d'arriver à la station terminale désirée. Ces stations intermédiaires représentent une partie des réseaux que les informations traversent avant d'arriver au destinataire.

Ceci nous amène au modèle complet de l'interconnexion de systèmes ouverts. (*fig. 2.4. Modèle complet de l'interconnexion de systèmes ouverts*)

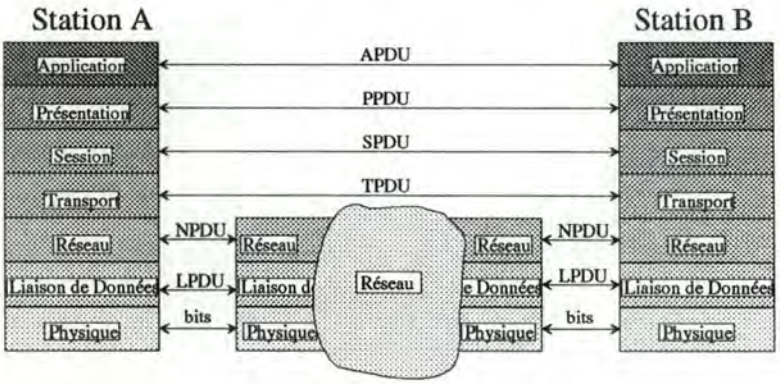


fig. 2.4. Modèle complet de l'interconnexion de systèmes ouverts

2.2. Présentation de MAP/MMS et de Mini-MAP

2.2.1. Présentation de MAP

Autrefois, lorsqu'une entreprise voulait automatiser un processus de fabrication, elle recherchait une solution sur mesure. Mais ces entreprises avaient de plus en plus de problèmes lorsqu'elles voulaient interconnecter leurs îles d'automation souvent hétérogènes, parce qu'elles venaient de producteurs différents. L'interconnexion de ces systèmes était certes possible, mais très chère. Sous la pression de la concurrence qui régnait sur le marché de l'automobile, General Motors (GM) décida de repenser sa structure d'automatisation. Et, au début des années '80, GM entama le développement d'une spécification d'un protocole pour l'interconnexion des appareils utilisés dans l'automatisation des processus de fabrication (MAP - *Manufacturing Automation Protocol*).

L'histoire du réseau local industriel MAP[3]

Sous la pression accrue de concurrents sur le marché de l'automobile, GM établit en 1980 un groupe de travail (*MAP Task Force*) dont le but était de développer pour l'environnement industriel un système de communication ouvert. Comme la création d'un tout nouveau système demande beaucoup d'expérience et un temps non négligeable, le groupe de travail décida de se baser sur des (quasi) standards définis par l'ISO dans le modèle en 7 couches (le *ISO Reference Model*).

Après 4 ans de travail, le groupe présenta en 1984 à la *National Computer Conference* (NCC), avec la collaboration des fournisseurs d'appareils d'automation les plus importants aux Etats Unis (tel que IBM, DEC, HP, ...), une première installation de test. Cet environnement de test était basé sur les couches 1 à 4 du modèle OSI. Le réseau par lequel passent toutes les informations est un réseau *Broadband** de type *token bus*.

Cette démonstration a prouvé que l'échange d'informations entre installations de provenance diverse était possible. Beaucoup de fabricants (tel Ford) ont été impressionnés par cette démonstration et ont commencé à réfléchir sur MAP, et

3 MAP est un réseau local industriel. La norme MAP 3.0 définit la pile de protocoles utilisés qui forment, eux, l'architecture de MAP. Dans la suite, nous parlerons donc de réseau MAP, d'architecture MAP et de norme ou de spécification MAP.

partout aux Etats Unis des groupes d'utilisateurs de MAP (*MAP User Groups*) se sont formés .

Les grandes entreprises européennes (telles Siemens) suivirent dès le début le phénomène MAP et leur intérêt pour MAP les incita à créer, en 1985, le "*European MAP User's Group*" (EMUG).

A l'Autofact en novembre 1985, GM présenta une autre installation déjà basée sur des produits prototypes existants. De plus, on illustra par cette installation la possibilité de réaliser un système de communication complet qui peut aussi entrer en communication avec d'autres réseaux tels que les réseaux utilisés dans les bureaux (des réseaux souvent de type Ethernet). L'installation réalisée était basée sur les couches 1 à 5 ainsi que sur la couche 7 du modèle OSI.

Le développement de MAP continua et en 1986 on présenta à Birmingham le premier réseau MAP : un réseau *Carrierband**. C'est aussi en 1986 que le principe du Mini-MAP et le protocole EPA firent leur apparition. Nous parlerons du Mini-MAP et de EPA au point 2.2.3.

La publication de la norme MAP 3.0 a été possible après la normalisation des couches supérieures du modèle OSI en 1988. Cette norme, basée sur les 7 couches du modèle OSI, utilise comme élément de service d'application majeur le MMS et est restée stable pendant quelques années.

Le vrai démarrage de MAP eut lieu en 1989 par la publication de la norme MAP 3.0. L'installation du premier réseau MAP 3.0 complètement fonctionnel en Europe fut réalisée fin '91 chez GM en Grande Bretagne[4]. Le tableau 2.2. donne un aperçu des étapes fondamentales de l'évolution du réseau local industriel MAP.

Simultanément au développement de MAP, il a fallu développer un protocole d'application adapté aux échanges d'informations entre applications dans un environnement d'automation. Un premier protocole utilisé était le SMF*, remplacé en 1984 par le MMFS* plus complet mais incompatible avec la notation ASN.1*.

On publia alors avec la norme MAP 3.0 le protocole d'application MMS compatible avec ASN.1 et devenu depuis lors un IS (*international standard*).

4 Référence /21/.

1980	Création de la MAP Task Force chez General Motors.
1984	Présentation de MAP 1.0 et SMF à la NCC à Las Vegas. L'installation de présentation se base sur les couches 1 à 4 et l'échange d'informations se fait sur un réseau *Broadband* à 5MBit/sec.
1985	Présentation de MAP 2.1 et de MMFS à l'Autofact à Detroit. On a implémenté les couches 1 à 5 et la couche 7 et on utilise des produits prototypes existants. On crée l'EMUG
1986	Présentation de MAP 2.2 à Birmingham avec cette fois-ci un réseau *Carrierband* à 5MBit/sec. En plus présentation du concept du Mini-MAP
1988	Présentation de MAP 3.0 à Baltimore et l'ISO accorde le statut de IS à MMS.
1989	Cette année présente le vrai début de MAP avec la publication de la norme MAP 3.0
1991	Le premier réseau MAP 3.0 complètement fonctionnel (en Europe) est installé.

tableau 2.2. Les étapes fondamentales de l'évolution de MAP

Architecture générale de MAP

MAP (*Manufacturing Automation Protocol*) est un réseau notamment local qui relie les différents équipements dans un environnement industriel. Nous avons déjà vu, lorsque nous avons parlé de l'histoire de MAP, que les protocoles de communications utilisés dans le réseau MAP sont choisis parmi les protocoles existants d'OSI. Ce n'est que pour la couche 7 (couche application) que le comité MAP a développé un protocole spécifique à la messagerie industrielle MMS, protocole devenu depuis lors une norme ISO (IS 9506 1/2) et qui peut être utilisé dans d'autres environnements si on le souhaite. L'architecture MAP peut se résumer par la figure 2.5. qui nous indique la pile des protocoles OSI (*OSI Protocol Stack*) utilisés avec MAP.

L'utilisation de ces standards permet d'être indépendant d'un producteur d'équipements particuliers et permet donc de relier des appareils venant de différents fabricants, ce que l'on désire souvent faire.

La couche physique de MAP représente la ligne physique et définit les caractéristiques physiques du médium de transmission. Cette ligne physique est le bus de transmission des données. Le comité MAP a choisi d'utiliser soit un réseau *Carrierband** sur un câble coaxial et qui permet une vitesse de transmission maximale de 5 Mbps, soit un réseau *Broadband** également sur câble coaxial mais

permettant cette fois-ci une vitesse de transmission maximale de 10 Mbps. Aujourd'hui, on rencontre en outre le cas où un réseau MAP à fibres optiques forme le *backbone* du système CIM de l'usine.

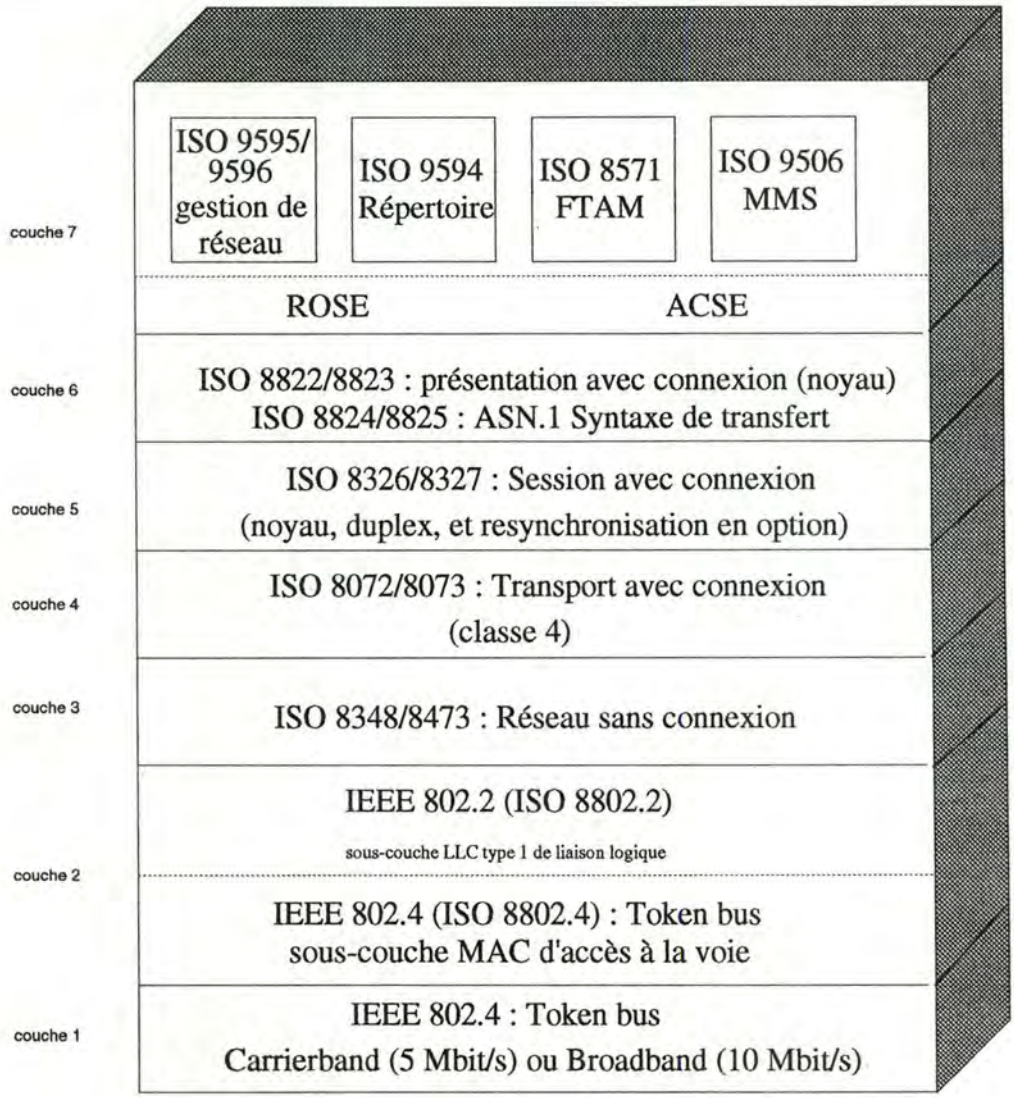


fig. 2.5. Pile de protocoles MAP

Pour le comité MAP, il existe principalement deux méthodes possibles pour accéder au bus. La première méthode est la méthode d'accès par compétition dit CSMA/CD*, souvent mieux connue sous le nom d'Ethernet. La seconde méthode est basée sur le principe d'un jeton (*token*) qui passe d'une station à une autre et partage de cette manière l'accès au bus. Seule la station qui possède le *token* peut en principe

accéder au bus. On parlera dans ce cas du *Token Bus*[5]. (*fig. 2.6. Principe du token bus*)

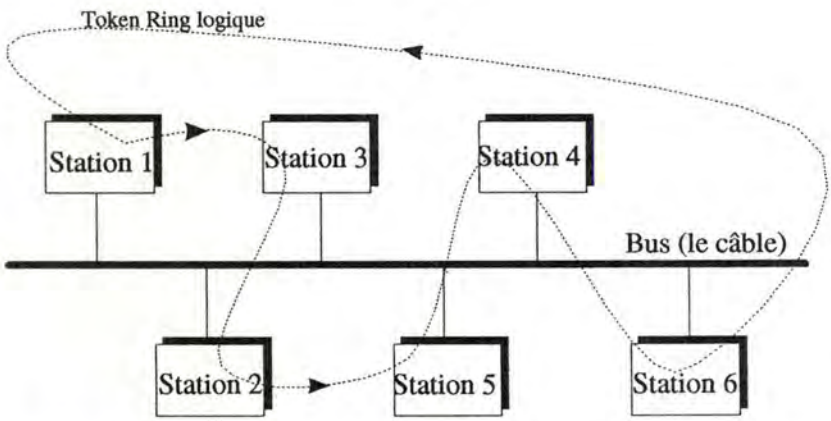


fig. 2.6. Principe du token bus (token ring logique)

Le grand avantage du réseau Ethernet c'est que dans beaucoup d'entreprises un réseau comparable est déjà installé principalement pour relier les ordinateurs des bureaux. Et on pourrait facilement élargir ce réseau pour couvrir toute l'usine.

Mais en fin de compte, pour le réseau MAP, c'est le *token bus*, qui l'a remporté. Pour l'utilisation dans un environnement industriel, le *token bus* offre quelques avantages que le CSMA/CD ne fournit que partiellement. En effet, le *token bus* permet un accès déterministe et sans conflit au bus. Cela signifie que chaque station située dans le *ring* logique recevra le *token* après un temps maximal défini et déterminé par le *token rotation time* (temps de rotation du *token*). L'accès au bus est évidemment sans conflit parce que seule la station qui possède le *token* peut envoyer quoi que ce soit sur le bus.

A ces deux avantages viennent s'ajouter deux autres. Premièrement, le principe du *token* permet de définir certaines priorités pour les stations. Ainsi une station peut, par exemple, tenir le *token* pendant un laps de temps plus important que d'autres stations. Le *token* bus offre d'autre part comme option la réponse immédiate[6] (*immediate response*) qui pourtant n'est pas utilisée dans MAP/MMS.

5 Le ***token bus*** est fonctionnelement un *token ring* réalisé sur un bus. De ce fait, on parlera aussi de *token ring* logique.

6 ***réponse immédiate*** : une station qui détient le *token* peut envoyer un message à une autre station qui ne fait pas nécessairement partie du *token ring* logique. Cette station peut alors répondre immédiatement (d'où le nom d'*immediate response*) au message que lui a adressé la première station sans qu'elle détienne le *token* ou qu'elle soit sur le *token ring* logique.

Par tous ces avantages, le *token bus* est bien adapté à l'environnement industriel où on exige en général des temps de réponse déterministes, ou même des réponses en temps réel.

Il est vrai que le *token ring* classique offre les mêmes avantages que le *token bus*, mais l'utilisation du *token ring* dans un environnement industriel est plus risqué. Si pour une raison quelconque, le *ring* est brisé, tout le réseau est inaccessible puisque le *token* et les informations ne peuvent plus passer. Dans le cas où un *token bus* serait coupé en deux, il y a toujours une partie du bus qui peut continuer à travailler.

C'est la partie inférieure de la couche liaison de données (la sous-couche MAC*) qui réalise cet accès au bus par la méthode du *token bus* (IEEE 802.4). La partie supérieure de la couche liaison de données est formée par la sous-couche LLC de liaison logique. La sous-couche LLC (*Logical Link Control*) de MAP est la version de type *connection less*[7] et elle échange par conséquent des données de type datagramme. Ce choix parait suffisant puisque la transmission sur un *token bus* dans un environnement local est assez stable et sûre.

Ce protocole nommé LLC[8] type 1 est bien adapté à l'environnement industriel parce qu'il est très facile à réaliser et assez rapide dans l'exécution. De plus, un protocole *connection less* se prête mieux à l'expédition de messages à un groupe ou à l'ensemble des stations connectées. L'utilisation d'un protocole orienté connexion n'est pas nécessaire à ce niveau car le comité MAP a décidé de choisir, comme on le verra plus tard, au niveau quatre un protocole de transport qui sera, lui, orienté connexion (*connection oriented transport service*). Un autre argument en faveur du LLC 1 est que c'est le standard pour Ethernet et que les services confirmés sont plus difficiles à réaliser.

En ce qui concerne la couche réseau (couche 3), le comité MAP a également opté pour un protocole *connection less*. Cette approche permet de simplifier l'interfaçage avec d'autres types de réseaux au niveau de la couche réseau. De plus, le mode *connection less* permet un achèvement rapide du service car il ne faut pas ouvrir et fermer une connexion à tous les coups.

---

7 ***connection oriented*** (orienté connexion) : pour ce mode de transmission il faut toujours d'abord ouvrir une connexion avant de pouvoir envoyer des données sur la ligne. Par après il faut bien sûr refermer la connexion. ***connection less*** (non orienté connexion) : ce mode de transmission permet d'envoyer des informations (des datagrammes) sans devoir ouvrir et fermer à chaque fois une connexion.

8 Il convient de rappeler ici que la sous-couche **LLC** de type 1 apporte le multiplexage.

La couche réseau de MAP correspond à la norme ISO 8473. Seul le modèle complet de cette norme est valable dans le cas de MAP[9].

Les couches inférieures de MAP sont donc de type *connection less*. Ce n'est qu'à partir de la couche 4, de laquelle nous parlerons maintenant, que le service devient un service orienté connexion (*connection oriented*). Ce sera par conséquent à la couche transport de jouer le rôle de charnière en offrant un service *connection oriented* à partir des services *connection less* fournis par les couches inférieures.

La couche transport offre, nous l'avons déjà précisé, un service fiable de bout en bout sur base d'un protocole orienté connexion ceci afin de débarrasser les couches supérieures des problèmes posés par des erreurs de transmission.

Comme la sous-couche LLC est de type 1 et donc d'une structure très simple, elle est incapable de procéder à une correction d'erreurs et à un contrôle de flux au niveau de la couche liaison de données. Pour cette raison, le comité MAP a choisi un protocole de transport de classe 4[10], défini dans ISO 8072/8073. Ce choix a aussi été nécessaire du fait que seule la classe 4 permet à la couche de transport de fournir un service *connection oriented* en utilisant les services *connection less* offerts par la couche réseau.

La couche session de MAP est basée sur le protocole de session avec connexion (ISO 8326/27). Cette couche ne joue ici qu'un rôle secondaire et pour l'utilisation courante on n'a besoin que de trois unités fonctionnelles : noyau, transmission duplex et parfois synchronisation. Le rôle de cette couche est minime puisque les échanges entre deux stations sont très simples et presque toujours de type requête/réponse dans le cadre d'une relation client/serveur.

De même, la couche présentation de MAP est également basée sur le protocole de présentation avec connexion. La seule unité fonctionnelle utilisée est le noyau. Cette simplicité de la couche de présentation est due au fait que, pour le réseau MAP, tous les éléments de service d'application (*application service element*) sont définis avec la notation ASN.1 (ISO 8824). La syntaxe de transfert est aussi définie

9 La norme prévoit encore deux autres possibilités : le sous-ensemble inactif (*inactive subset*) et le sous-ensemble sans segmentation (*non segmentation subset*).

10 Dans la spécification de la couche transport (ISO 8072/73), l'ISO définit 5 classes de transport. La classe 4 est la classe la plus complexe. Cette classe a été désignée pour l'utilisation sur des réseaux de type C - réseau avec un taux d'erreurs signalées et non signalées élevé.

par les règles de codage ASN.1 (ISO 8825) et la couche de présentation est quasi transparente.

La couche d'application enfin utilise évidemment toujours l'élément de service commun ACSE (*Association Control Service Element*) pour établir et pour clôturer une association d'application.

Le service de base de la couche d'application est la messagerie industrielle assurée par l'élément de service d'application MMS sur lequel nous reviendrons plus en détail au point 2.2.2.. MMS joue un rôle fondamental pour l'interfonctionnement des équipements industriels. MMS offre en outre des services élémentaires de chargement de fichiers.

Pour les applications qui nécessitent des services plus évolués de transfert de fichiers, MAP offre la possibilité d'utiliser l'élément de service d'application FTAM* (*File Transfer, Access and Management*).

Ces éléments de service de base sont complétés par un service de répertoire (ISO 9594) et par un service de gestion de réseau MAP. Ces deux services exigent la mise en oeuvre d'opérations distantes, définies par l'élément de service commun ROSE (*Remote Operations Service Element*).

Avec l'arrivée de la version 3.0 de MAP, un nouvel élément apparaît dans la spécification. On a introduit une facilité supplémentaire qui favorise, d'une part, la portabilité de logiciels (des applications) et, d'autre part, l'indépendance des logiciels vis-à-vis d'un changement de protocole. Cette nouveauté permet aussi d'accéder plus facilement aux services offerts par MAP sans connaître tous les détails des protocoles utilisés.

Cette interface a été définie par le comité MAP comme une interface d'application (*Application Programming Interface* - API) pour les éléments de service MMS et FTAM. L'utilisateur de MMS ou/et de FTAM n'est pas obligé d'utiliser les API, mais l'utilisation de celui-ci facilite le travail avec MMS et FTAM. Ceci d'autant plus que ces interfaces sont facilement programmables avec des langages de programmation courants tel le langage 'C', par exemple.

L'idée de l'API est de fournir aux applications une interface de haut niveau (*High Level Programming Interface*). L'application accède à cette interface de haut niveau. L'API analyse la demande venant de l'application et transforme cette demande en plusieurs services de bas niveau. Ces services de bas niveau correspondent aux services offerts par MMS ou par FTAM. L'application est donc

totalement indépendante des services de bas niveau. Les figures 2.7. et 2.8. montrent respectivement la localisation de l'API et la réaction à une demande de service de haut niveau émanant de l'application.

Pour illustrer ce principe d'interface d'application prenons un exemple. Lorsqu'on veut charger un domaine dans un serveur, 3 services MMS sont impliqués dans le processus : *Initiate Download Sequence*, *Download Segment* et *Terminate Download*. Pour lancer le chargement, l'API offre un service ***File Download***.

Au moment, où l'application fait appel à ce service de haut niveau, l'API se charge de surveiller le bon déroulement de l'exécution de ce service. D'abord, l'API fait appel au service MMS confirmé *Initiate Download Sequence* qui initialise le chargement d'un domaine dans un serveur. Si la réponse à ce service est négative, l'API donne une information appropriée à l'application. Si elle est positive, l'API attend un *Download Segment*.ind et répond en envoyant un segment du domaine.

Supposons à présent que tous les segments ont été chargés. L'API attend alors une indication de fin de chargement du domaine. Après avoir reçu cette indication, l'API y répond et donne également une confirmation à l'application.

Il est possible de réaliser immédiatement dans l'application elle-même ce que nous venons d'expliquer. Mais en faisant appel aux services de l'API, on laisse à l'API le soin de contrôler et de réaliser une opération complexe telle que le chargement d'un domaine.

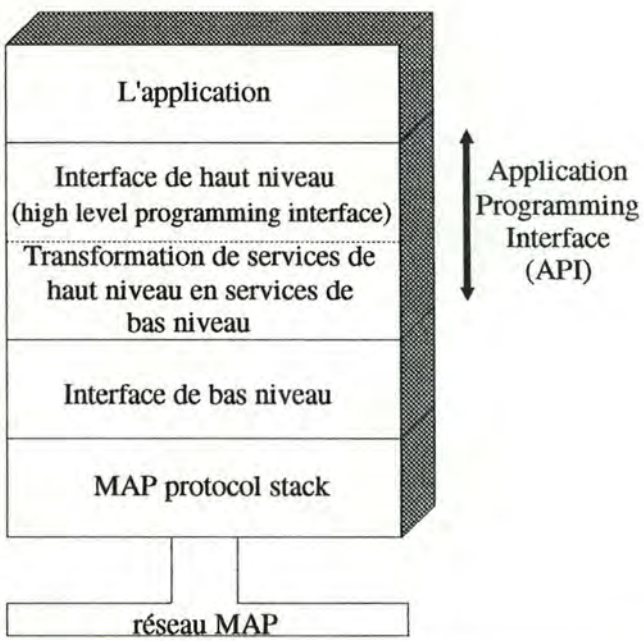


fig. 2.7. Localisation de l'API

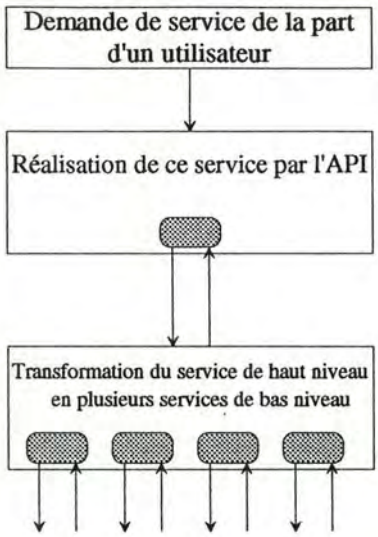


fig. 2.8. Réaction à une demande de service

C'est ainsi que se termine la présentation de l'architecture de MAP qui définit la pile des protocoles OSI constituant le réseau MAP. Pour être complet, il faudrait encore préciser en détail les options retenues pour chaque protocole et les limitations pratiques imposées sur les valeurs et les différents paramètres. Nous ne le ferons pas ici. Ces informations se trouvent dans la spécification MAP 3.0 ou dans le livre de H. Nussbaumer (réf. /13/).

Le point sur lequel nous nous pencherons à présent est le MMS, qui constitue l'élément de service d'application le plus important de MAP.

2.2.2. La messagerie industrielle (MMS)

Les principes

La messagerie industrielle MMS (*Manufacturing Message Specification* - ISO 9506) est, comme nous l'avons déjà dit, un élément de service d'application et se situe par conséquent au niveau de la couche 7 du modèle OSI. En fournissant des services normalisés qui couvrent les fonctions courantes exécutées dans un environnement industriel, MMS facilite la communication entre équipements de fabrication provenant de fabriquants différents.

Ceci montre clairement que MMS joue un rôle fondamental dans le réseau MAP. C'est le MMS qui assure l'interfonctionnement d'équipements divers et c'est le reste du réseau MAP qui sert principalement à assurer l'interconnexion.

Se pose alors la question suivante : comment est-il possible de fournir un standard qui assure l'interfonctionnement d'équipements si ces équipements peuvent être de types différents?

Il est absolument clair que des équipements différents (composés d'une partie *hardware* et d'une partie *software*) sont impliqués dans un processus de fabrication. Mais chaque équipement contient, du côté *software,* des éléments de même type (tels que des variables, par exemple) qui seront manipulés de la même manière. C'est sur cette idée que se base MMS. (Mais, comme MMS est prévu pour une multitude d'appareils, on n'a pas toujours les mêmes fonctionnalités pour tous.)

Par ailleurs, MMS permet la définition de standards d'accompagnement (*Companion Standards*) qui permettent d'adapter ou de compléter, selon des règles très strictes, MMS aux besoins d'appareils spécifiques tels des SPS, NC*/DNC, des robots, etc..

A la fin de cette brève introduction à MMS, on peut citer quelques points caractéristiques du MMS.

- ❑ MMS est indépendant du *hardware*.
- ❑ MMS est très flexible grâce à l'utilisation possible et prévue des standards d'accompagnement.
- ❑ La sémantique de MMS est définie sur base d'objets.
- ❑ La syntaxe de MMS est définie en ASN.1.
- ❑ MMS travaille en mode *connection oriented.*
- ❑ L'idée du MMS est importante même au delà de MAP (on retrouve ses concepts dans les réseaux PROFIBUS*, SINEC*, etc..)
- ❑ MMS utilise l'ACSE et les services offerts par la couche présentation, mais n'utilise pas le ROSE[11].

Après cette présentation générale de l'élément de service d'application MMS, les paragraphes suivants permettront de regarder d'un peu plus près le MMS. Nous en présenterons les concepts fondamentaux et les services. Le protocole n'est pas abordé ici, mais ceux qui s'y intéressent peuvent trouver des informations détaillées dans ISO 9506/2.

11 L'utilisation du **ROSE** serait logique dans le cadre de MMS puisque un des concepts fondamentaux de MMS est le modèle client serveur. ROSE offre en fait des services ou un *invoker* lance une opération chez un *performer* qui le renseigne sur le résultat de l'opération. Des noms couramment employés pour l'*invoker* et le *performer* sont respectivement client et serveur. (réf. /2/)

Les concepts fondamentaux

La messagerie industrielle MMS se base sur trois concepts fondamentaux :

① le modèle client/serveur
② le modèle de VMD (*Virtual Manufacturing Device*)
③ le modèle d'objet

Nous expliquerons un à un les modèles de base de MMS afin de pouvoir nous baser sur ces concepts lorsque nous présenterons les différents services offerts par MMS.

① LE MODÈLE CLIENT/SERVEUR

A la base de tous les services MMS se situe un modèle tout simple, le modèle client/serveur dont le fonctionnement est le suivant : un appareil, dénommé client, demande un service à un deuxième appareil, dénommé serveur. Pour demander le service, le client envoie cette demande de service au serveur par le réseau MAP. Le serveur fait de son mieux pour exécuter la demande et renvoie le résultat au client par l'intermédiaire du réseau MAP. La figure 2.9. reprend schématiquement ce modèle de client/serveur en représentant un client qui envoie une demande (une requête) pour recevoir une réponse de la part du serveur. Ce modèle général peut, dans l'interaction avec les autres modèles, fonctionner parfois autrement. Il se peut que ce soit le serveur qui est l'initiateur.

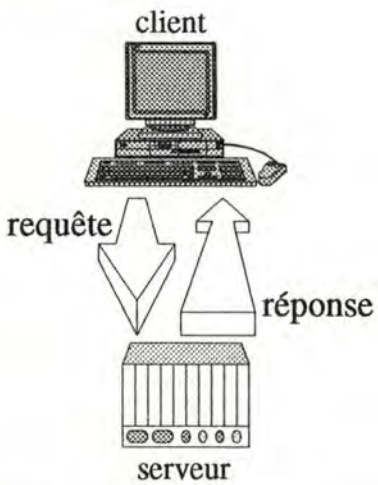


fig. 2.9. Schéma du modèle client/serveur dans MMS

Précisons un peu et expliquons plus en détail le fonctionnement. Le client (demandeur de service) demande un service auprès d'un serveur. Pour ce faire, la demande de service passe par le *stack* MAP, traverse le réseau MAP et arrive auprès du serveur comme l'indication d'une demande de service de la part d'un client. Le serveur essaie de répondre à la demande et envoie la réponse qui passe à son tour par le réseau MAP pour arriver chez le client comme confirmation de la demande. (*fig. 2.10. Fonctionnement du modèle client/serveur*)

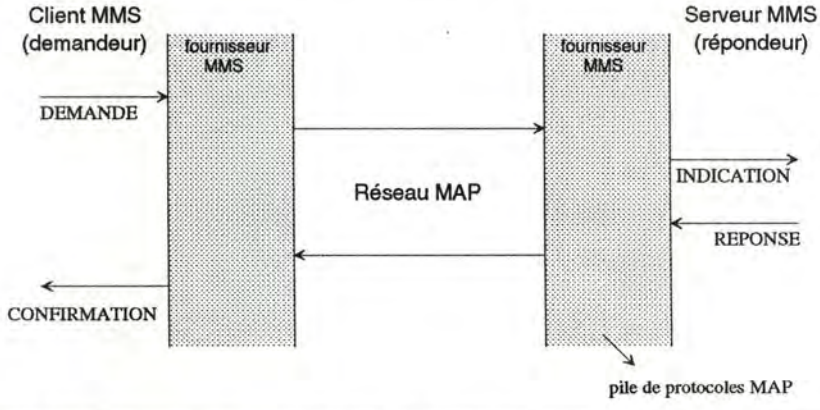


fig. 2.10. Fonctionnement du modèle client/serveur.

Comme la plupart des échanges qui se font entre deux équipements sont de type requête/réponse, on peut dire que ce modèle est parfaitement adapté aux besoins d'une messagerie industrielle. Un exemple type que l'on trouve dans tout environnement industriel est la demande de lecture d'une variable de la part du client qui reçoit pour réponse du serveur la valeur de la variable. De plus, toutes les commandes des machines de fabrication peuvent être considérées comme des services vis-à-vis des ordinateurs d'atelier.

② LE MODÈLE DE VMD (*virtual manufacturing device*)

Des équipements de tout genre peuvent être connectés à un réseau MAP. Afin de représenter de manière uniforme le comportement externe de ces équipements de fabrication et pour contourner en conséquence le problème de la diversité des équipements, la norme MAP définit le modèle de VMD (*Virtual Manufacturing Device* - machine virtuelle de fabrication).

En décrivant pour l'extérieur une structure et un comportement virtuel, la machine virtuelle de fabrication représente les ressources et les fonctions d'un

appareil réel. Cette structure et ce comportement virtuel sont identiques pour tous les équipements. Mais pour chaque équipement de type différent, il est évident qu'il faut avoir une VMD appropriée, puisque c'est la VMD qui joue le rôle d'interface entre la machine réelle et sa représentation à l'extérieur. (*fig. 2.11. Le principe de la VMD*)

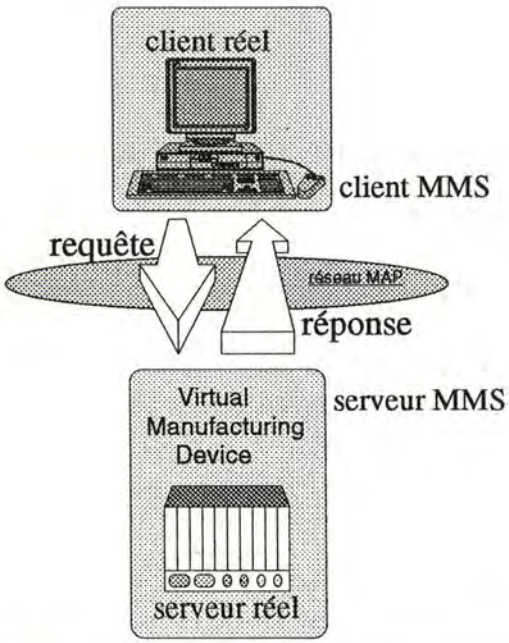


fig. 2.11. Le principe de la VMD

D'une manière générale, on peut établir une analogie entre la VMD et un driver de périphérique en ce sens que la VMD offre une interface formelle pour la mise en oeuvre de la machine réelle, en se limitant aux seules fonctions qui ont trait à la communication.

L'idée de VMD est étroitement liée à la notion de serveur. En effet, le modèle client/serveur que nous venons d'expliquer se base sur le fait qu'un client demande un service à un serveur. On standardise l'interface du serveur vers le client pour que le client n'ait pas besoin de connaître le détail de la réalisation sur le serveur. Grâce à une telle interface, on fait abstraction du comportement réel du serveur. Chaque équipement peut ainsi répondre à une même demande de la part d'un client sans que le client doive se préoccuper du fait qu'il communique avec des équipements de type différent. Cette interface est donc nécessairement du côté du serveur. Pour tout type d'équipement et pour chaque constructeur, il existe une interface différente puisque cette interface doit modéliser le comportement réel du serveur. On trouve ainsi, par

exemple, des interfaces pour les automates programmables, pour les commandes numériques ou pour les commandes de robots. Un client communique donc via cette interface avec l'équipement réel. La norme MMS décrit la structure générale de cette interface qu'est la VMD.

La figure 2.12. illustre ce modèle de VMD en montrant un client - dans ce cas ci un contrôleur de cellule - qui peut communiquer avec 3 équipements de type différent : un automate programmable, une commande numérique et une commande de robots.

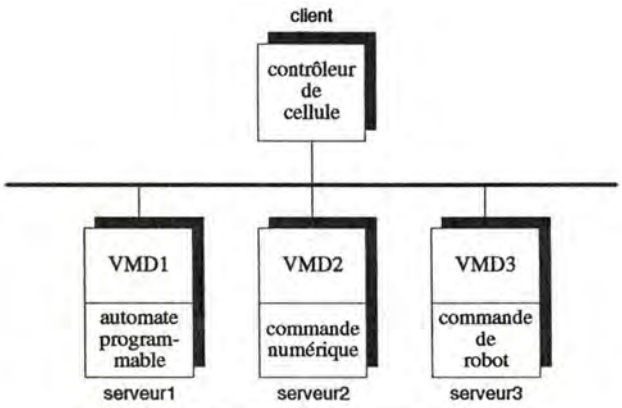


fig. 2.12. Illustration du modèle de VMD

Un équipement ne peut donc jouer le rôle de serveur que lorsqu'il dispose au moins d'une VMD. Un client peut ne disposer d'aucune VMD, ou en posséder plusieurs. Si un équipement ne dispose d'aucune VMD, il ne peut être que client. Une station peut, le cas échéant, être client et serveur en même temps.

Pour bien situer la place de la VMD dans le modèle OSI, il est utile de se rappeler la structure de la couche d'application sans pour autant entrer trop dans les détails. Pour ce faire, il faut introduire le concept de processus d'application (*application process*-AP). (*fig.2.13. Application process*) Le processus d'application se compose de deux parties : la première partie que nous présentons brièvement est l'agent d'application (*application agent*-AA) qui joue le rôle d'interface du *stack* OSI avec l'utilisateur et les ressources disponibles. L'agent d'application ne fait pas partie intégrante du modèle OSI puisqu'il est spécifique à chaque machine. La seconde partie du processus d'application est constituée de l'entité d'application (*application entity*-AE) qui répond aux besoins de communication dans et avec le modèle OSI. C'est l'entité d'application qui fait partie du modèle OSI.

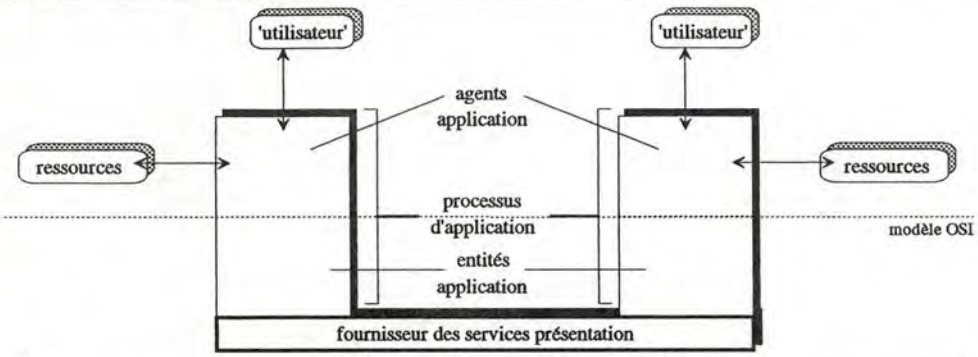


fig. 2.13. Application process (réf. /6/)

Comme la VMD concerne la communication, il faut que chaque VMD intègre au moins une AE pour permettre cette communication. Chaque AE ne peut correspondre qu'à une seule VMD. L'accès à la VMD (du niveau couches OSI) se fait par des PSAP (*Presentation Service Access Points*). (*fig. 2.14. Processus d'application d'un serveur MMS*)

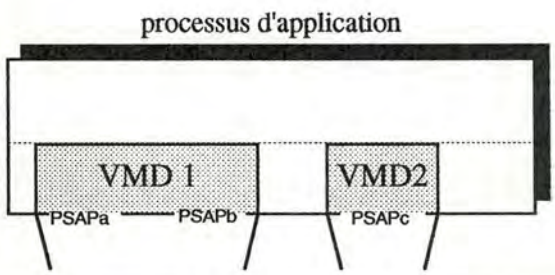


fig. 2.14. Processus d'application d'un serveur MMS

La VMD est un objet et on peut accéder à l'objet VMD à partir de services déterminés. La VMD peut contenir d'autres objets tels que des domaines par exemples. Le modèle d'objet est expliqué dans le paragraphe suivant.

③ LE MODÈLE D'OBJET

L'objet est le concept central de la messagerie industrielle. Tout le modèle MMS est décrit en termes d'objets définis par leurs caractéristiques et par les opérations qu'on peut effectuer sur ces objets. Le fonctionnement de MMS est complètement défini par la norme sous forme de manipulations d'objets situés dans la VMD comme on l'a vu au point précédent. Par les services, le client accède à ces objets situés dans la VMD d'un serveur. (fig. *2.15. Principe object user/object provider*)

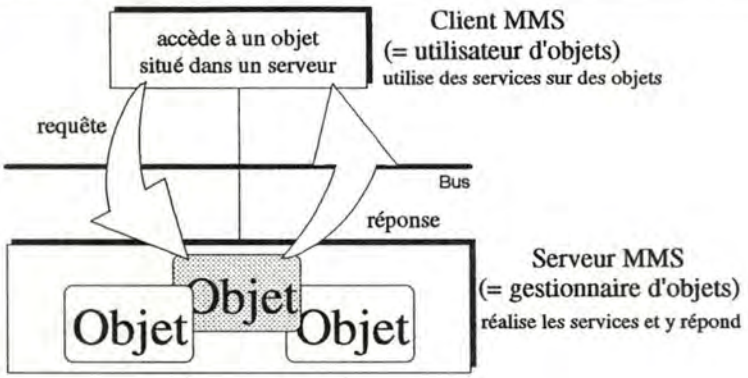


fig. 2.15. Principe object user/object provider

Chaque objet est une instance particulière d'une classe d'objets définie par la norme MMS. La norme MMS distingue les objets nommés et les objets anonymes qui, eux, ne sont accessibles que par leur adresse. Le tableau 2.3. et le tableau 2.4. reprennent respectivement la liste des 12 objets nommés et la liste des 4 objets anonymes en indiquant chaque fois le nom anglais de l'objet et le correspondant français.

Nom anglais	Nom français
Named variable	Variable nommée
Scattered access	Accès dispersé
Named variable list	Liste nommée de variables
Named type	Type nommé
Semaphore	Sémaphore
Event condition	Condition événementielle
Event action	Action événementielle
Event enrollment	Enregistrement d'événements
Journal	Journal
Domain	Domaine
Program invocation	Instance de programme
Operator station	Station opérateur

tableau 2.3. classes des objets nommés

Nom anglais	Nom français
Transaction	Transaction
Upload state machine	Automate de sauvegarde
Unnamed variable	Variable anonyme
Semaphore entry	Rubrique de sémaphore

tableau 2.4. classes des objets anonymes

Un objet nommé se caractérise non seulement par son nom mais encore par sa portée (*scope*)[12] qui représente la zone de visibilité de l'objet. Cette portée ne peut pas être changée par un service MMS.

Chaque objet peut être prédéfini ou être créé et détruit en cours de route. Ces derniers objets sont de type dynamique. Les objets prédéfinis sont le plus souvent de type statique.

La durée de vie d'un objet est limitée par la durée de vie du VMD auquel l'objet appartient. Un objet disparaît donc lorsque son *scope* disparaît. Mais on peut aussi demander explicitement la destruction d'un objet par un service MMS. Cette possibilité d'effacement d'un objet n'est en général possible que pour les objets dynamiques.

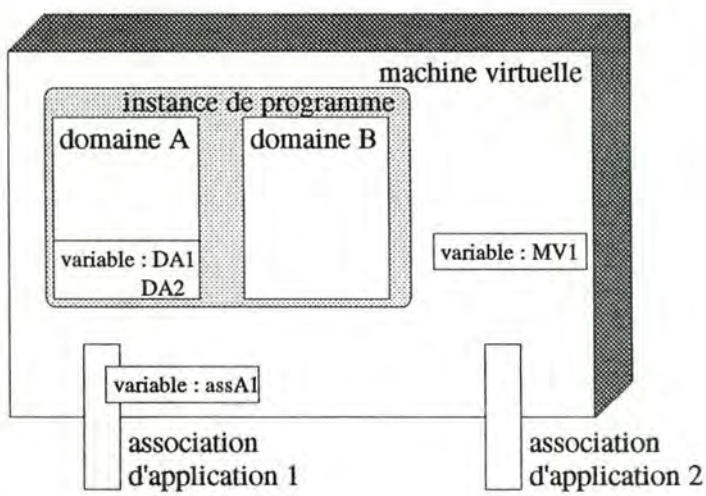


fig. 2.16. La portée d'un objet

En guise d'illustration de ce qui vient d'être dit sur la portée d'un objet, la figure 2.16. montre un exemple schématique qui se base sur l'objet 'variable'. La variable assA1 n'est accessible que pour l'association d'application A. La portée de la variable MV1 est la machine virtuelle et la portée des variables DA1 et DA2 est le domaine A. En dehors de leur portée, les variables sont invisibles.

Il est encore important de dire que chaque objet peut se trouver dans différents états. Ces états et la transition entre les états sont repris dans des diagrammes états-transitions qui figurent dans la norme MMS (ISO 9506 1/2).

12 Trois portées sont définies dans MMS : portée limitée à la VMD, au domaine ou à une association d'application.

On trouvera une présentation et une explication plus détaillée des différents objets MMS dans la section suivante, section consacrée aux services MMS.

Les services offerts par MMS

La norme MMS définit 79 services[13] pour manipuler les objets (également définis dans MMS). En pratique, un équipement réel ne fournira naturellement qu'un nombre limité de services. Ainsi, les applications réelles ne nécessitent souvent qu'un sous-ensemble des classes d'objets et des services disponibles. L'objectif poursuivi par la définition des 79 services différents est d'offrir un ensemble de services généralement nécessaires et utiles dans l'environnement industriel. Mais comme une norme ne peut pas couvrir toutes les exceptions et tous les cas particuliers possibles, on peut élargir les services par les standards d'accompagnements spécifiés pour différents équipements tels que les automates programmables.

Il est commode de regrouper les services MMS en ensembles fonctionnels qui rassemblent des services apparentés. MMS offre à l'utilisateur 10 ensembles fonctionnels repris au tableau 2.5.. On peut déjà noter ici que l'ensemble fonctionnel de gestion générale de l'environnement MMS est toujours présent, soit partiellement, soit entièrement, car c'est l'élément fonctionnel de base qui rassemble les services nécessaires à l'établissement et à la fermeture de la communication entre utilisateurs MMS.

Dans le regroupement en ensembles fonctionnels, nous nous sommes fixés sur les objets MMS qui sont manipulés par les services. Tous les services qui traitent un même objet sont regroupés dans un ensemble.

13 La norme MMS définit 79 services 'normaux'. A ces 79 services viennent s'ajouter des services de gestion de fichiers et des 'modificateurs'.

Nom anglais	Nom français
Environment and general management	Gestion générale de l'environnement MMS
VMD support	Gestion de la VMD
Domain management	Gestion des domaines
Program invocation management	Gestion des instances de programme
Variable access	Accès aux variables
Semaphore management	Gestion des sémaphores
Operator communication	Communication avec l'opérateur
Event management	Gestion des événements
Journal management	Gestion de journal
File management	Gestion des fichiers

tableau 2.5. Les ensembles fonctionnels de services MMS

Chaque accès à un équipement réel se fait à partir d'un des groupes de services qui agit sur les objets. La figure 2.17. donne une visualisation de l'accès au système réel à partir d'une couche virtuelle qui entoure la machine réelle. Cette couche virtuelle est formée des 10 ensembles fonctionnels de services MMS.

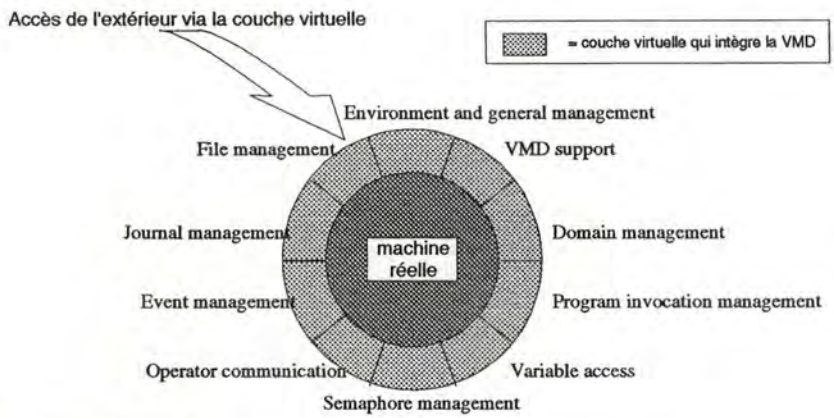


fig. 2.17. L'accès à la machine réelle

Parlons maintenant des différents ensembles fonctionnels, des services offerts par les ensembles et des classes d'objets concernées. (Nous utilisons pour les titres les noms anglais des ensembles fonctionnels, alors que dans le texte nous employons la dénomination française.)

ENVIRONMENT AND GENERAL MANAGEMENT

L'ensemble fonctionnel de gestion générale de l'environnement MMS (*Environment and general management*) est le seul ensemble fonctionnel qui doive

obligatoirement être présent partiellement ou entièrement. En effet il est nécessaire pour l'établissement et la fermeture d'une association entre deux applications.

Par conséquent, cet ensemble fonctionnel offre des services qui permettent de demander

- l'établissement de la communication et l'initialisation de l'environnement MMS (*Initiate*).
- la terminaison ordonnée de la communication (*Conclude*).
- l'abandon de la communication (*Abort*).
- l'annulation d'un service en cours (*Cancel*).
- la notification des erreurs de protocole (*Reject*).

(*fig. 2.18. L'ensemble fonctionnel 'Environment and general management'*)

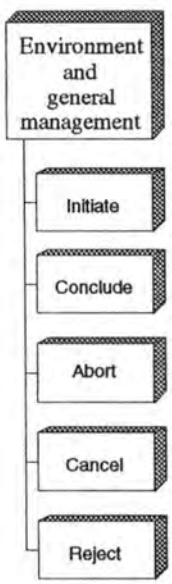


fig. 2.18. L'ensemble fonctionnel 'Environment and general management'

VMD SUPPORT

L'ensemble fonctionnel de gestion de la VMD (*VMD Support*) offre des services qui permettent d'obtenir des informations sur une VMD.

Nous avons déjà vu le concept de VMD dans la section précédente. Pour cette raison nous ne nous attarderons pas à le réexpliquer. Mais ce que nous voulons faire ici, c'est rappeler l'importance de la VMD pour la réalisation d'un travail sur des équipements répartis et interconnectés.

En effet, pour permettre une communication efficace entre les différents équipements, il est nécessaire que l'accès aux équipements soit standardisé de

manière à ce que le comportement réel se cache derrière une apparence virtuelle. C'est exactement la VMD qui décrit ce comportement virtuel en utilisant les objets définis dans MMS. La VMD n'est que du côté des serveurs puisque les objets se trouvent aussi de ce côté. Ceci ne signifie en rien que le client ne doive pas connaître les objets. Il est évident que le client ne dispose pas du modèle de VMD (sauf s'il est en même temps un serveur) et qu'il ne possède pas d'objets MMS. Mais il doit, néanmoins, connaître en détail les différents objets offerts, puisque c'est lui qui va activer les services qui manipulent les objets MMS.

L'ensemble fonctionnel de gestion de la VMD offre des services pour

- demander l'état de la VMD (*Status*).
- signaler spontanément l'état de la VMD (*Unsolicited Status*).
- obtenir le nom des objets de la VMD (*Get Name List*).
- identifier la VMD (*Identify*).
- changer le nom des objets de la VMD (*Rename*).
- obtenir la liste des capacités (ressources) de la VMD (*Get Capability List*).

(*fig. 2.19. L'ensemble fonctionnel 'VMD support'*)

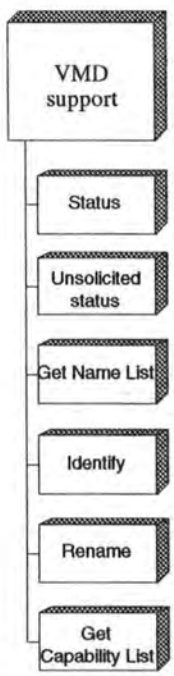


fig. 2.19. L'ensemble fonctionnel 'VMD support'

On constate que la VMD support n'offre aucun service pour créer ou détruire la VMD.

DOMAIN MANAGEMENT

L'élément fonctionnel de gestion des domaines (*domain management*) offre des services de chargement, de sauvegarde et de gestion de domaines. Tous les services portent donc sur une instance de la classe d'objets domaines.

Un objet domaine représente un sous-ensemble des ressources de la VMD et, plus précisément, une partie de mémoire d'un équipement d'automation où on peut charger des données de tout genre. Cette possibilité permet de créer et de charger les données indépendamment des routines qui les traitent. On répond ainsi particulièrement bien aux besoins actuels de l'automation parce qu'on peut facilement faire tourner un même programme avec des données différentes. Ceci permet en plus l'utilisation des mêmes équipements pour fabriquer différents produits (en chargeant un autre programme et/ou d'autres données).

Par l'objet domaine, le MMS définit une vue virtuelle des données chargées. Cette vue est indépendante de la manière selon laquelle les données existent réellement (sur disque, en mémoire, ...). Pour l'utilisateur, seuls les services et les opérations sur les domaines sont connus. L'utilisateur ne peut donc en aucune manière accéder directement aux données réelles.

Ce qui est important ici c'est le comportement du domaine et non son contenu. Le contenu doit être discuté entre les applications client et serveur.

Prenons deux cas types de l'automation afin de visualiser ce concept de domaine que nous venons d'expliquer.

cas 1 : contrôleur NC (*numerical control*)

Un contrôleur NC est prévu pour pouvoir réaliser différentes pièces. Pour ce faire, on charge un programme.

Mais pour fabriquer la pièce, le contrôleur NC a encore besoin de données sur la pièce à fabriquer. Ces données peuvent être de type différent telles que les données de la pièce proprement dite, les données de départ, les données de correction, etc.. Les données nécessaires sont donc également chargées.

Chaque programme et chaque donnée forment un domaine particulier. (*fig. 2.20. Représentation symbolique des domaines de l'exemple*). C'est le

programme qui utilise les données, contenues dans les domaines. Ceci pour fabriquer la pièce tout en respectant les contraintes fixées par ces données.

En chargeant d'autres données, il est facile de produire une autre pièce ou la même pièce avec d'autres données de correction. On voit donc clairement l'avantage de la modularité offerte.

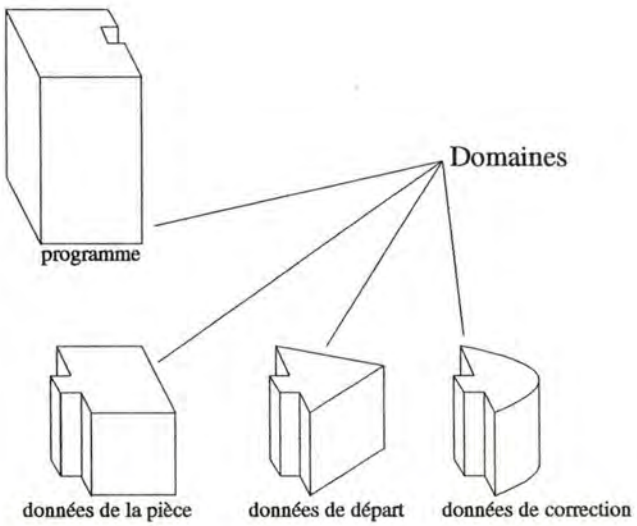


fig. 2.20. Représentation symbolique des domaines de l'exemple

Ceci ressemble fortement au concept d'ordinateur sur lequel tourne un programme qui utilise différentes données. Mais dans MMS on fait abstraction de l'emplacement réel des données et du programme, puisque tout est contenu dans un domaine. Nous étudierons la gestion des instances de programmes, un peu plus loin, dans ce chapitre.

cas 2 : l'automate programmable

Les différentes fonctions d'un automate programmable sont un autre exemple possible de domaines. La figure 2.21. montre des fonctions comme exemple de domaines. Ces fonctions peuvent être regroupées en un module exécutable ce qui, en MMS, correspond à une instance de programme. Nous parlerons plus en détail des instances de programmes, un peu plus loin.

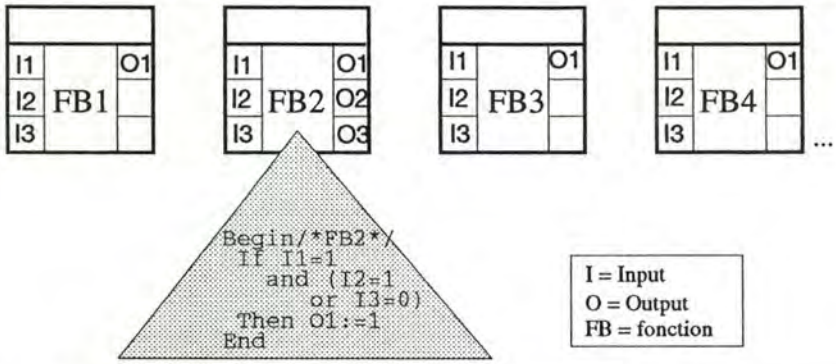


fig. 2.21. Schéma des fonctions d'un automate programmable

Le premier service offert par l'ensemble fonctionnel de gestion des domaines est celui du chargement d'un domaine. Il permet à un client d'initier la création d'un nouveau domaine sur un serveur. Pour ce faire, le client utilise le service *Initiate Download Sequence*. Ce sera alors le serveur qui contrôlera le chargement des différents segments du domaine par *Download Segment*. Après avoir terminé le chargement du domaine, le serveur indique la fin de l'opération au client par *Terminate Download Sequence*. Cette suite de demandes de service est représentée dans la figure 2.22.[14]

On voit que le chargement d'un domaine peut se faire par plusieurs segments. Ceci est seulement dû au fait qu'on ne veut pas transmettre des unités de données trop importantes. Ce chargement par segment ne peut en aucun cas être interprété comme offrant la possibilité de charger partiellement un domaine.

Le deuxième service est celui de la sauvegarde qui est juste la contrepartie du précédent. Il permet de transférer le contenu d'un domaine d'un serveur vers le client. Ici aussi, c'est le client qui lance les services. Les services utilisés pour réaliser la sauvegarde sont *Initiate Upload Sequence*, *Upload Segment* et *Terminate Upload Sequence*.

14 Dans l'exemple, les segments sont chargés à partir du client puisque celui-ci joue également le rôle de serveur de fichier. Mais rien ne dit que cela doive toujours être le cas.

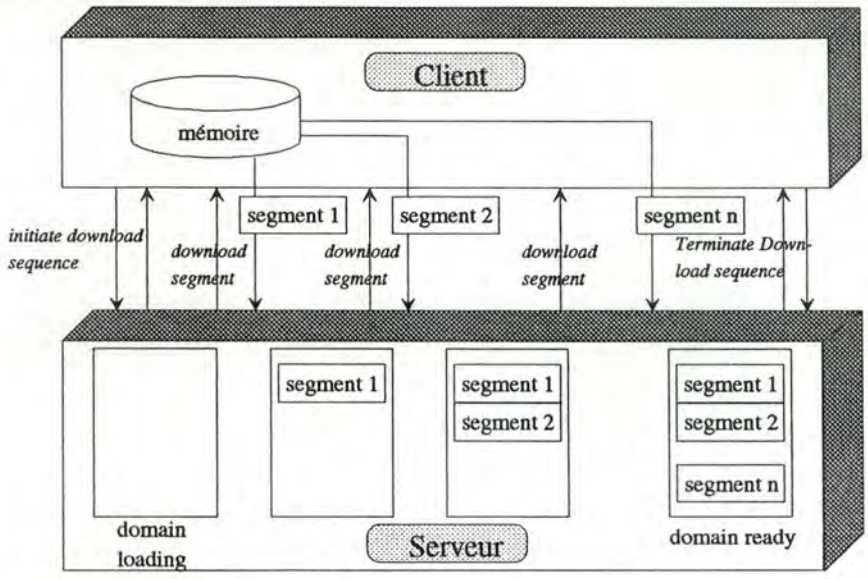


fig. 2.22. Evolution dans le temps du chargement d'un domaine

Les services de téléchargement et de télésauvegarde fonctionnent en principe de la même façon que les services de chargement et de sauvegarde que nous venons d'expliquer. Mais cette fois-ci, le serveur charge le domaine non à partir du client, mais à partir d'une troisième station qui joue alors le rôle de serveur de fichiers. C'est le client qui demande le téléchargement auprès d'un tiers (*third party*) par *Load Domain Content.* Dans le cas d'une télésauvegarde, le client utilise le service *Store Domain Content.* Il est également possible que le serveur charge le domaine à partir d'une mémoire interne au serveur.

Si le serveur éprouve le besoin de charger ou de sauvegarder un domaine, il peut le dire au client par les services *Request Domain Download/Request Domain Upload.* Ce sera alors de nouveau le client qui initiera le chargement/la sauvegarde du domaine.

Pour compléter les services de base de chargement et de sauvegarde, MMS offre un service de lecture d'attributs du domaine (*Get Domain Attributes*) et de purge du domaine (*Delete Domain*). (*fig. 2.23. L'ensemble fonctionnel 'Domain management'.*)

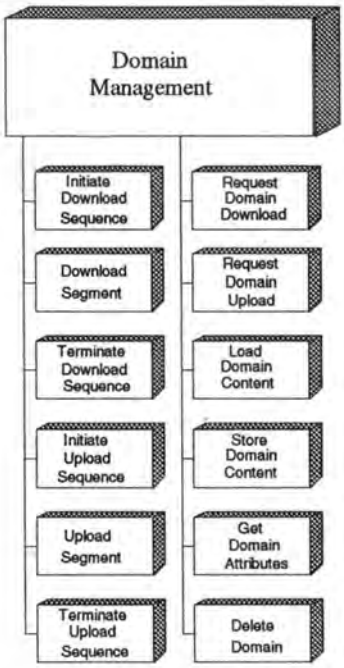


fig. 2.23. L'ensemble fonctionnel 'Domain management'

PROGRAM INVOCATION MANAGEMENT

L'ensemble fonctionnel de gestion des instances de programme (*program invocation management*) offre des services de création, de lancement, de fin et de suppression d'instances de programmes. Une instance de programme de MMS peut être considérée comme une tâche (par exemple, un *task* dans un environnement *multi-tasking*) qui exécute le code d'un programme contenu dans un ou plusieurs domaines. L'instance de programme est un objet MMS lié à un ou à plusieurs domaines. Si une instance de programme relie des domaines, le contenu des domaines détermine complètement l'interaction entre ces domaines.

Pour clarifier un peu cette idée d'instance de programme, nous rappellerons les 2 cas cités pour la gestion des domaines. Dans le premier cas, (contrôleur NC), il y a moyen de créer une instance de programme à partir des 4 domaines. Cette instance de programme peut alors être exécutée par le service de lancement (*fig. 2.24. instances de programme dans le cas du contrôleur NC*). Et ce sera l'information contenue dans le domaine 'programme' qui déterminera l'utilisation des domaines 'données'.

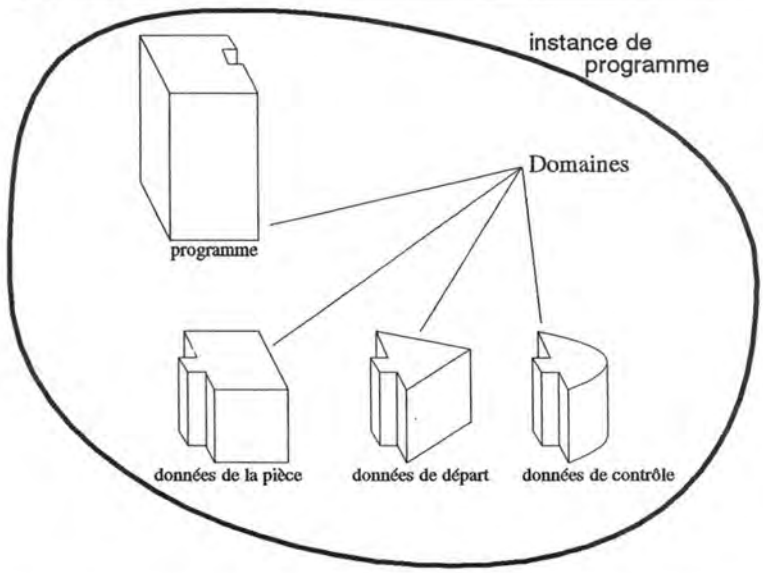


fig. 2.24. Instance de programme dans le cas du contrôleur NC

Le cas de l'automate programmable va dans la même direction. Les différentes entités fonctionnelles peuvent être regroupées en un module exécutable qui, lui, est vu comme une instance de programme.

MMS offre plusieurs services principaux de gestion d'instances de programmes:

- le service de création (*Create Program Invocation*)
- le service de suppression (*Delete Program Invocation*)
- le service de lancement (*Start*)
- le service de suspension (*Stop*)
- le service de reprise (*Resume*)
- le service de réinitialisation (*Reset*)
- le service de destruction (*Kill*)

A ces services de base s'ajoute un service de lecture des attributs de l'instance de programme (*Get Program Invocation Attributes*). (*fig. 2.25. l'ensemble fonctionnel 'Program invocation management'*)

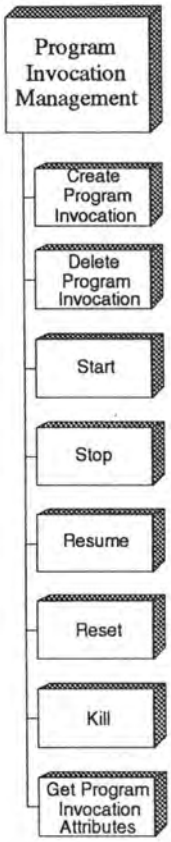


fig. 2.25. L'ensemble fonctionnel 'Program invocation management'

VARIABLE ACCESS

L'ensemble fonctionnel d'accès aux variables (*variable access*) offre, comme son nom l'indique, des services qui permettent d'accéder aux variables. A ces services s'ajoutent des services de création et de destruction de variables.

Une variable peut être, par exemple, la représentation d'une valeur fournie par un capteur. Pour MMS, la notion de variable a un sens plus large et englobe aussi d'autres objets tels que des paramètres de configuration.

Les seules variables auxquelles nous nous intéressons ici, sont les variables qui ont trait à la communication et qui se situent de ce fait dans la VMD. L'accès à ces variables s'effectue par l'intermédiaire d'objets abstraits (les variables MMS).

La norme MMS définit 5 objets abstraits d'accès aux variables :

1. La ***variable anonyme*** est directement en correspondance avec une zone de mémoire définie seulement et de façon permanente par une adresse. Elle ne

peut donc ni être créée ni être détruite. Dans la terminologie MMS, l'accès a une variable est dit public (*Public Access Method*) pour indiquer que l'accès se fait par l'intermédiaire d'une adresse.

2. La ***variable nommée*** est obtenue en donnant un nom à une variable. Ceci est utile, d'une part, pour rendre l'accès plus commode et, d'autre part, pour être indépendant de l'adresse mémoire réellement occupée. Si la variable nommée désigne une variable de la VMD qui n'est accessible que par l'adresse, l'accès est également dit public. Mais si la variable nommée permet l'accès à une variable nommée de la VMD, l'accès est dit privé.
3. La ***variable à accès dispersé*** est un objet composé, défini pour faciliter l'interfonctionnement avec des équipements anciens où les variables ont souvent toutes la même longueur (p.ex. 16bits). Ceci exclurait pratiquement l'utilisation de variables de type tableau ou structure. Dans ce cas, il est commode de regrouper des variables au sein d'une variable à accès dispersé. Cette variable peut être créée et éventuellement détruite par le client.
4. Le type abstrait, ***type nommé,*** a été introduit pour faciliter la manipulation des types en leur attribuant un nom. Ceci permet de faire référence au type par son nom et plus par sa description détaillée.
5. Le type abstrait ***liste nommée de variables*** est un deuxième objet qui se compose de variables anonymes, nommées et/ou à accès dispersé. L'objet a été introduit pour faciliter la manipulation simultanée de plusieurs variables, tout en conservant l'individualité des variables.

MMS offre 11 types de variables simples et 2 types de variables composées qui figurent au tableau 2.6.

Type	Description
BOOLEAN	Booléen. Dimension implicite
BIT STRING	Chaîne de bits.
INTEGER	Entier signé
UNSIGNED	Entier non signé
FLOATING POINT	Valeur flottante (type 'real')
REAL	Réel (ISO 8824)
OCTET STRING	Chaîne de bytes
VISIBLE STRING	Chaîne de caractères imprimables
GENERALIZED TIME	Date et heure (imprimables)
BINARY TIME	Date et heure (compactées en binaire)
BCD	Valeur décimale codée en binaire
ARRAY	Tableau (éléments de même type)
STRUCTURE	Structure (types différents)

tableau 2.6. Les types de variables offerts par MMS

Afin de diminuer les problèmes d'incohérence des données, une transmission d'une valeur d'une variable ne peut être qu'une réussite ou un échec, et rien d'autre. En outre, l'accès à une variable doit être atomique, c.-à.-d. sans interruption (si possible).

Comme nous l'avons déjà mentionné précédemment, MMS offre des services qui permettent de manipuler des variables.

- Lecture de la valeur de variables (*Read*).
- Notification au client de la valeur d'une variable (*Information Report*).
- Ecriture de la valeur d'une variable (*Write*)
- Création de variables et de types nommés (*Define Named Variable, Define Scattered Access, Define Named Variable List, Define Named Type*)
- Suppression de variables et de types nommés (*Delete Variable Access, Delete Named Variable List, Delete Named Type*)
- Lecture d'attributs de variables et de types nommés (*Get Variables Access Attributes, Get Scattered Access Attributes, Get Named Variable List Attributes, Get Named Type Attributes*)

(*fig. 2.26. L'ensemble fonctionnel 'Variable access'*)

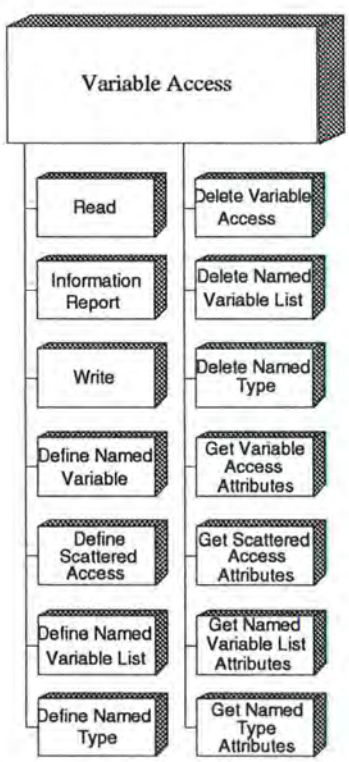


fig. 2.26. L'ensemble fonctionnel 'Variable access'

SEMAPHORE MANAGEMENT

L'ensemble fonctionnel de gestion des sémaphores (*semaphore management*) permet aux clients MMS de contrôler, de synchroniser et de coordonner l'utilisation de ressources partagées entre plusieurs clients. Les services offerts correspondent en fait à des fonctions d'un système d'exploitation réparti et font appel à l'objet sémaphore.

Classiquement, le sémaphore est un objet qui comprend une variable S (indiquant le nombre de droits d'utilisation) et une file d'attente des tâches. Le sémaphore MMS correspond à cette définition classique du sémaphore.

Dans MMS, l'objet sémaphore est un objet qui contient un nombre de jetons et une file d'attente. Chaque jeton représente un droit d'utilisation. Un client ne peut utiliser la ressource (critique) que lorsqu'il possède un droit d'utilisation, c.-à.-d. lorsqu'il est en possession d'un jeton.

Deux classes de sémaphores sont distinguées dans MMS : le *token semaphore* (sémaphore banalisé) et le *pool semaphore* (sémaphore étiqueté).

Le sémaphore banalisé sert spécifiquement à synchroniser les applications entre utilisateurs MMS. Les jetons ne possèdent aucune individualité. Quant au sémaphore étiqueté, il est utilisé pour contrôler l'utilisation des ressources précises. Là, on a donc besoin de nommer les jetons.

La figure 2.27. montre l'exemple d'un serveur d'imprimante contrôlé par un sémaphore banalisé. Cinq imprimantes du même type sont connectées au serveur d'imprimante. Par conséquent, 5 clients peuvent imprimer en même temps sans devoir indiquer l'imprimante sur laquelle ils veulent sortir leurs données. Un sixième arrivant se verra attribuer la première imprimante libre.

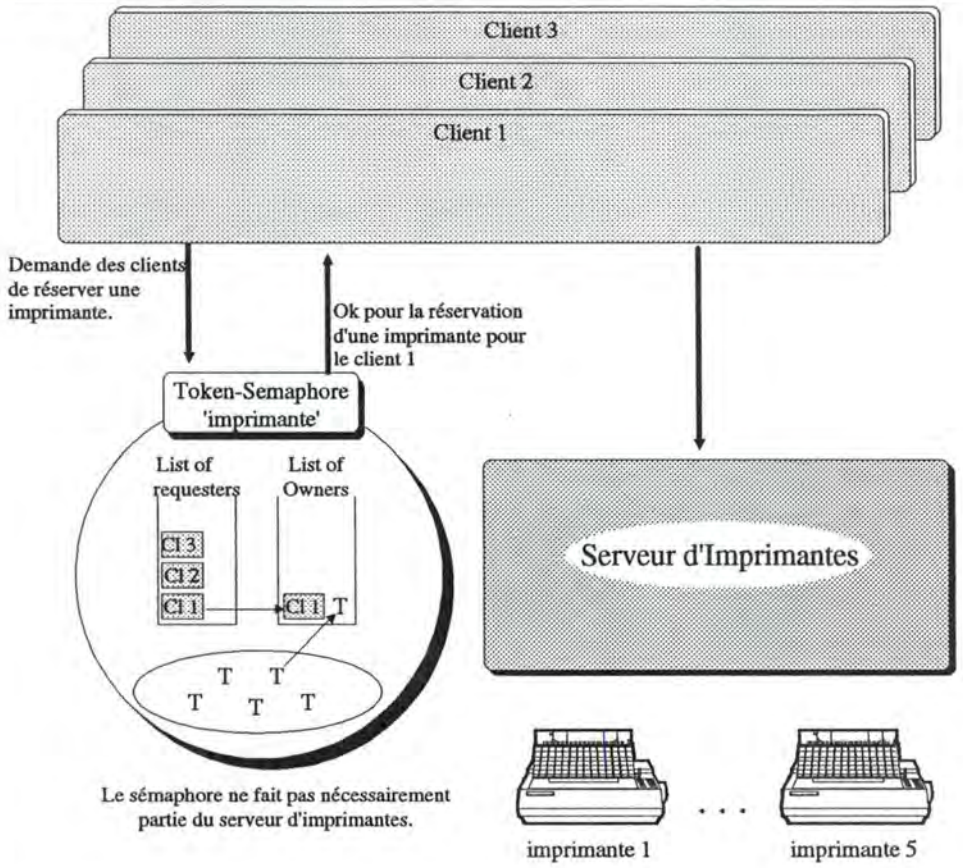


fig. 2.27. Mécanisme de sémaphore banalisé

La figure 2.28. montre, par contre, l'exemple d'un sémaphore étiqueté avec des jetons nommés. Des imprimantes de différents types (laser, jet d'encre, matricielle, postscript, ...) sont connectées au serveur d'imprimantes. Le serveur peut choisir l'imprimante sur laquelle il désire imprimer ses données. Si l'imprimante souhaitée n'est pas libre, l'impression n'est pas immédiate, même si les quatre autres imprimantes sont libres.

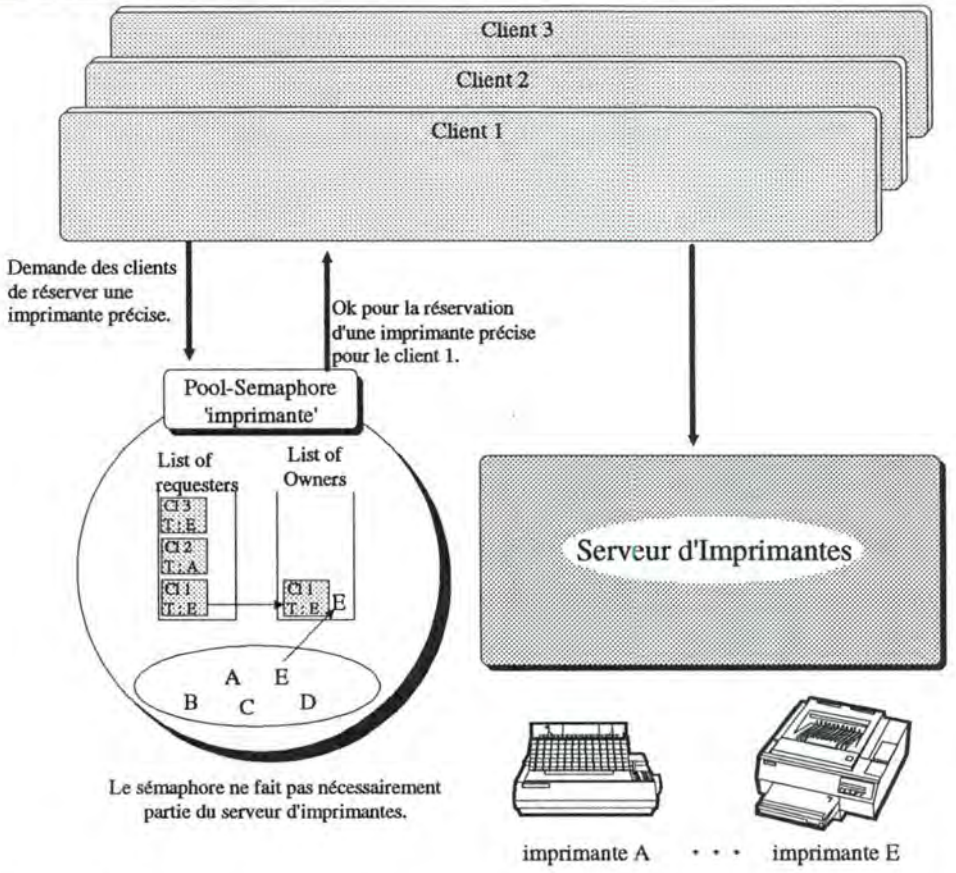


fig. 2.28. Mécanisme de sémaphore étiqueté

A côté de l'objet sémaphore, MMS définit encore l'objet *semaphore entry* (rubrique de sémaphore). L'objet rubrique de sémaphore joue un rôle similaire à un descripteur de tâche pour un sémaphore classique. La rubrique de sémaphore contient toutes les informations utiles pour le sémaphore, telles que l'identification du demandeur, l'application utilisatrice et l'association d'application donnée. L'objet rubrique de sémaphore est explicitement créé sur demande d'un client.

MMS offre des services de gestion des sémaphores pour :

- prendre le contrôle d'un sémaphore (*Take Control*).
- libérer un sémaphore (*Relinquish Control*).
- créer un sémaphore banalisé (*Define Semaphore*).
- détruire un sémaphore banalisé (*Delete Semaphore*).
- lire l'état d'un sémaphore, de sémaphores nommés et de rubriques de sémaphore (*Report Semaphore Status, Report Pool Semaphore Status, Report Semaphore Entry Status*).

(*fig. 2.29. L'ensemble fonctionnel 'Semaphore management'*)

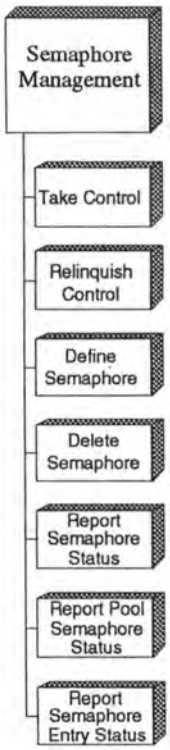


fig. 2.29. L'ensemble fonctionnel 'Semaphore management'

OPERATOR COMMUNICATION

L'ensemble fonctionnel de communication avec l'opérateur (*operator communication*) offre des services d'entrée et de sortie avec des équipements qui jouent le rôle de terminaux vis-à-vis du système.

Le modèle ISO prévoit pour ce type de communication l'élément de service d'application terminal virtuel (*virtual terminal*-VT/ISO 9040/9041). Mais cette solution, bien que prévue par MAP, est souvent trop complexe pour l'environnement industriel.

La communication avec l'opérateur s'effectue dans MMS via l'objet *Operator Station* qui donne une vue abstraite de la station opérateur réelle. Cette procédure facilite l'implémentation des opérations d'entrée et de sortie. L'objet station opérateur fait partie de la VMD.

MMS offre des services pour

- lire un message fourni par la station opérateur (*INPUT*).
- afficher un message sur la station opérateur (*OUTPUT*).

(*fig. 2.30. L'ensemble fonctionnel 'Operator communication'*)

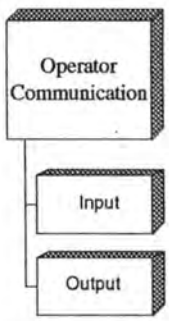


fig. 2.30. L'ensemble fonctionnel 'Operator communication'

C'est le client MMS qui demande la lecture d'une ligne au terminal. Si plusieurs lignes doivent être lues, plusieurs demandes doivent être lancées. Par conséquent, on ne peut entrer un message que lorsqu'on a été invité à le faire. Le client MMS peut aussi demander l'affichage pur et simple d'un ensemble de lignes. La figure 2.31. montre un déroulement possible de demande de lecture et de demande d'affichage.

[Une station opérateur est, soit uniquement destinée à l'*Input* (le clavier), soit uniquement destinée à l'*Output* (l'écran), soit les deux en même temps (l'écran tactile).]

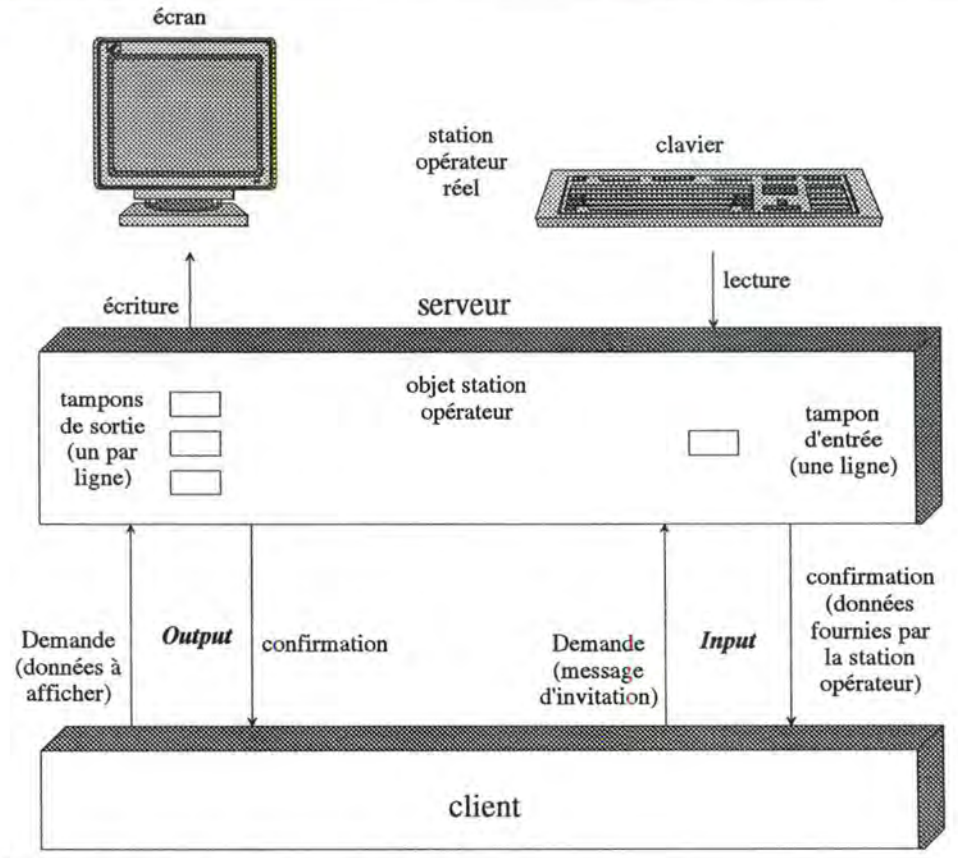


fig. 2.31. Demande de lecture et d'écriture

EVENT MANAGEMENT

Le concept d'événement est fortement lié aux systèmes répartis. Le travail en cours est interrompu parce qu'un événement externe ou interne à l'équipement est survenu. Il s'agit alors de réagir à cet événement.

Pour faire abstraction des événements réels et des réactions aux événements réels, la norme MMS ajoute 3 objets à la VMD :

1. *Event Condition* - EC (condition événementielle)
2. *Event Action* - EA (action événementielle)
3. *Event Enrollment* - EE (enregistrement d'événements)

MMS distingue deux classes d'événements : d'une part, l'événement peut être déclenché par un client (événement déclenché - *Network Trigger Event*) ce qui est

montré à la figure 2.32.; d'autre part, l'événement peut être interne au serveur (p. ex. une interruption (*interrupt*)). Cette interruption survient en dehors de la VMD. Pour la faire entrer dans le cadre MMS, on associe une variable booléenne à chaque événement possible. Cette liste de variables est parcourue périodiquement (*fig. 2.33. Evénement interne au serveur*)

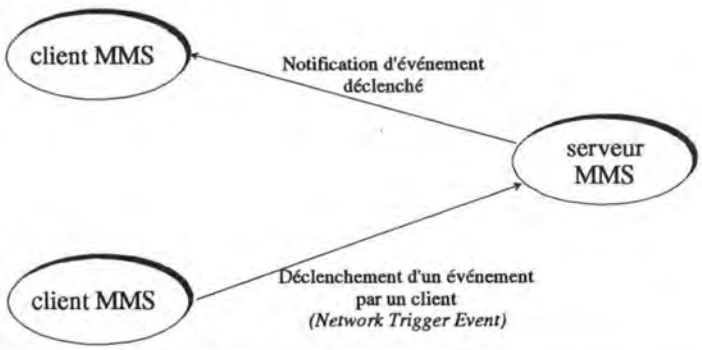


fig. 2.32. Evénement déclenché par un client.

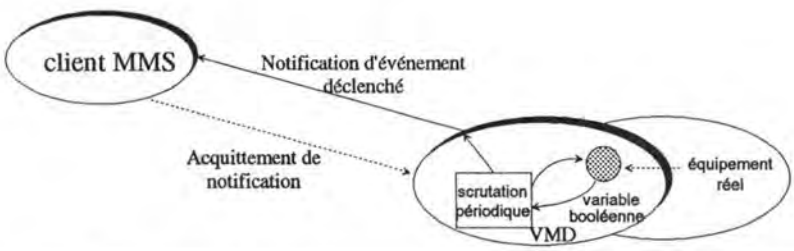


fig. 2.33. Evénement interne au serveur

Lorsqu'un événement est survenu, il est notifié à un client (*Event Notification*), après avoir peut-être effectué une action définie au préalable dans l'objet EA.

L'autre moyen d'annoncer l'arrivée d'un événement se fait par la confirmation d'un service attaché à l'événement (par *Attach To Event Condition*). Quand l'événement survient, il est nécessaire de déterminer à quel service confirmé il est attaché. Après avoir accompli le service, on envoie la confirmation qui est en même temps la notification. (*fig. 2.34. Service attaché à un événement*)

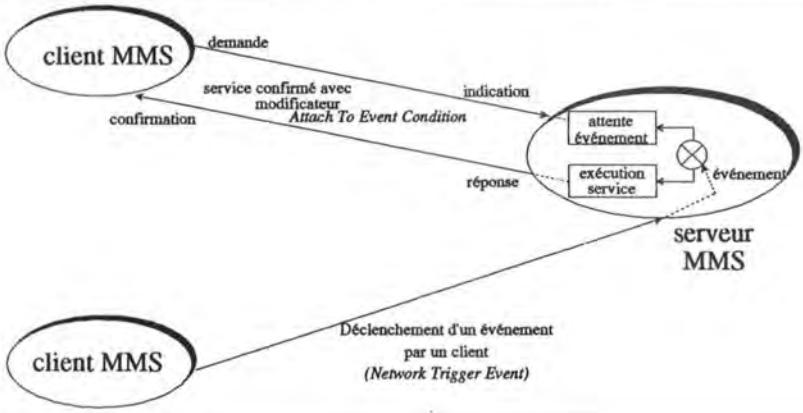


fig. 2.34. Service attaché à un événement

L'information nécessaire pour retrouver le client auquel il faut notifier l'événement se trouve dans l'objet EE. Plusieurs EE attachés à une seule condition événementielle permettent de notifier l'événement à plusieurs clients en 'même' temps.

MMS offre des services de

- notification d'un événement (*Event Notification, Acknowledge Event Notification*).
- modification d'attributs d'une EC et de déclenchement d'une EC déclenchée (*Alter Event Condition Monitoring, Trigger Event*)
- gestion des EC (*Define Event Condition, Delete Event Condition, Get Event Condition Attributes, Report Event Condition Status*)
- gestion des EE (*Define Event Enrollment, Delete Event Enrollment, Get Event Enrollment Attributes*)
- lecture d'état et de modification d'un EE (*Report Event Enrollment Status, Alter Event Enrollment*)
- gestion des EA (*Define Event Action, Delete Event Action, Get Event Action Attributes, Report Event Action Status*)
- lecture des récapitulatifs d'alarme (*Get Alarm Summary, Get Alarm Enrollment Summary*)

(*fig. 2.35. L'ensemble fonctionnel 'Event management'*)

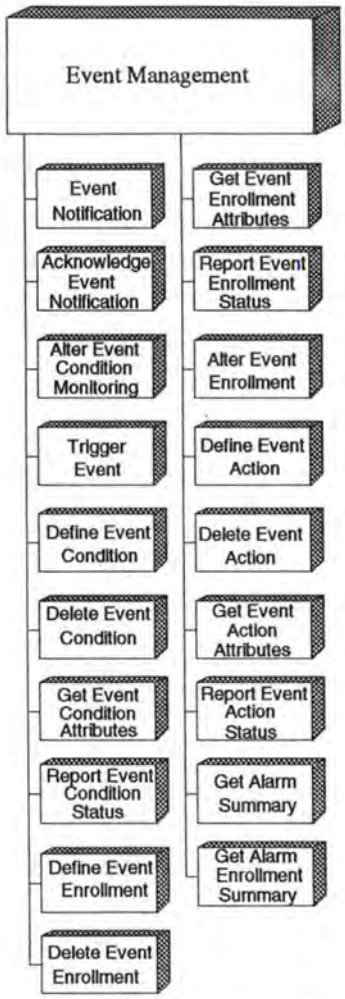


fig. 2.35. L'ensemble fonctionnel 'Event management'

JOURNAL MANAGEMENT

L'ensemble fonctionnel de gestion de journal (*journal management*) offre des services destiné à manipuler et à modifier un journal.

Un journal est un ensemble d'enregistrements qui constituent une trace du fonctionnement du système. Dans MMS, il s'agit d'une trace commandée par un client pour être conforme au modèle client/serveur[15]. Par cette méthode, le client peut insérer ce qu'il veut dans le journal. Le journal correspond donc alors à un ensemble d'informations fournies par le client.

15 L'autre méthode consiste en une trace automatique, où le serveur écrit lui-même des enregistrements selon les conditions fixées au préalable par le client.

Deux nouveaux objets sont définis au sein de la VMD : l'objet journal (*journal*) et l'objet rubrique de journal (*journal entry*). Chaque journal peut avoir un nombre 'illimité' de rubriques de journal (*fig. 2.36. Le journal et les rubriques de journal*).

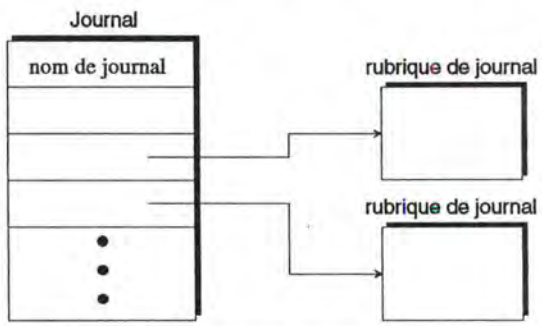


fig. 2.36. Le journal et les rubriques de journal.

MMS offre des services pour

- lire des rubriques (*Read Journal*).
- créer des rubriques (*Write Journal*).
- réinitialiser un journal (*Initialize Journal*).
- créer un journal (*Create Journal*).
- supprimer un journal (*Delete Journal*).

(*fig. 2.37. L'ensemble fonctionnel 'Journal management'*)

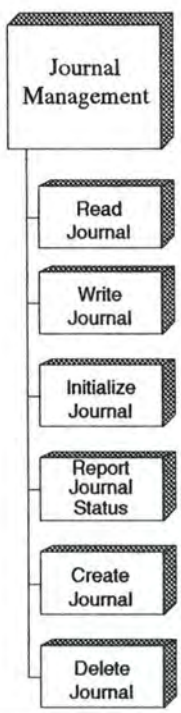


fig. 2.37. L'ensemble fonctionnel 'Journal management'

FILE ACCESS, FILE MANAGEMENT

La norme MAP recommande FTAM*16 (*File Transfer, Access and Management*) comme outil de gestion des fichiers. Pour les applications simples, souvent rencontrées dans l'environnement industriel, MMS offre des services élémentaires de gestion de fichier qui n'ont pas pour autant valeur de norme. C'est pour cette raison, que nous ne ferons que les citer. (*fig. 2.38. Les services d'accès aux fichiers et de gestion des fichiers*)

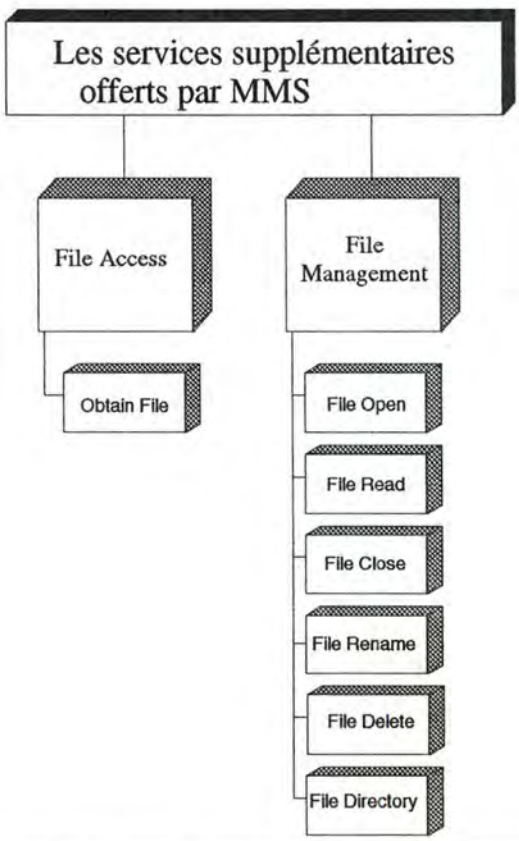


fig. 2.38. Les services d'accès aux fichiers et de gestion des fichiers.

A côté des services, on trouve dans la norme MMS la notion de 'modificateur' (*modifier*). Nous avons déjà rencontré ce concept lors de l'explication du *Event*

16 Pour plus de détails sur FTAM, nous conseillons de lire référence /6/.

Management. Là nous avons parlé du *Attach To Event Condition* qui attache l'exécution d'une demande de service confirmé à la réalisation d'un événement.

L'autre modificateur rencontré dans la norme MMS se situe dans l'ensemble fonctionnel de gestion de sémaphores. Il s'agit du *Attach To Semaphore.*

L'idée des modificateurs est de pouvoir retarder l'exécution d'un quelconque service confirmé jusqu'à ce qu'une certaine condition soit remplie. Ce service est alors conditionné de telle manière qu'il n'est exécuté que lorsque l'événement auquel il est lié survient ou lorsque le client obtient le *token.*

C'est ainsi que se termine la présentation de MMS. On trouvera à l'annexe A une liste de la totalité des services MMS ainsi qu'une brève explication.

Nous abordons à présent le deuxième standard international qui est MAP/EPA et les stations Mini-MAP.

2.2.3. Présentation de MAP/EPA et de Mini-MAP

Le chapitre 2.2.1. a présenté l'architecture MAP complète qui incorpore les 7 couches du modèle OSI. Cette architecture permet de réaliser des réseaux d'usines évolués. Mais, à des niveaux plus bas, plus près de la production proprement dite, les exigences sont autres que celles du niveau de l'usine. Ainsi, dans les cellules à portée limitée, on exige souvent des temps de réponse brefs. De plus, les messages qui circulent sont de petite taille (15 à 20 byte d'information utile). Par ailleurs, le nombre de cellules et donc le nombre de stations ayant de fortes contraintes en termes de temps et de taille est très important. On exige par conséquent un prix peu élevé pour ces équipements.

2 contraintes sont primordiales : - le temps de réponse
- le prix (le coût)

Or, l'architecture MAP est assez complexe et de ce fait coûteuse. De plus, comme l'architecture MAP complète se base sur l'ensemble des 7 couches du modèle OSI, le temps d'exécution est généralement incompatible avec les objectifs fixés. En

d'autres termes, bien que l'architecture MAP complète soit bien adaptée au niveau de l'usine, elle est souvent inadaptée aux niveaux inférieurs.

Pour pallier ces 'faiblesses' de MAP, on a introduit dans la norme MAP l'architecture EPA (*Enhanced Performance Architecture*) montrée à la figure 2.39..

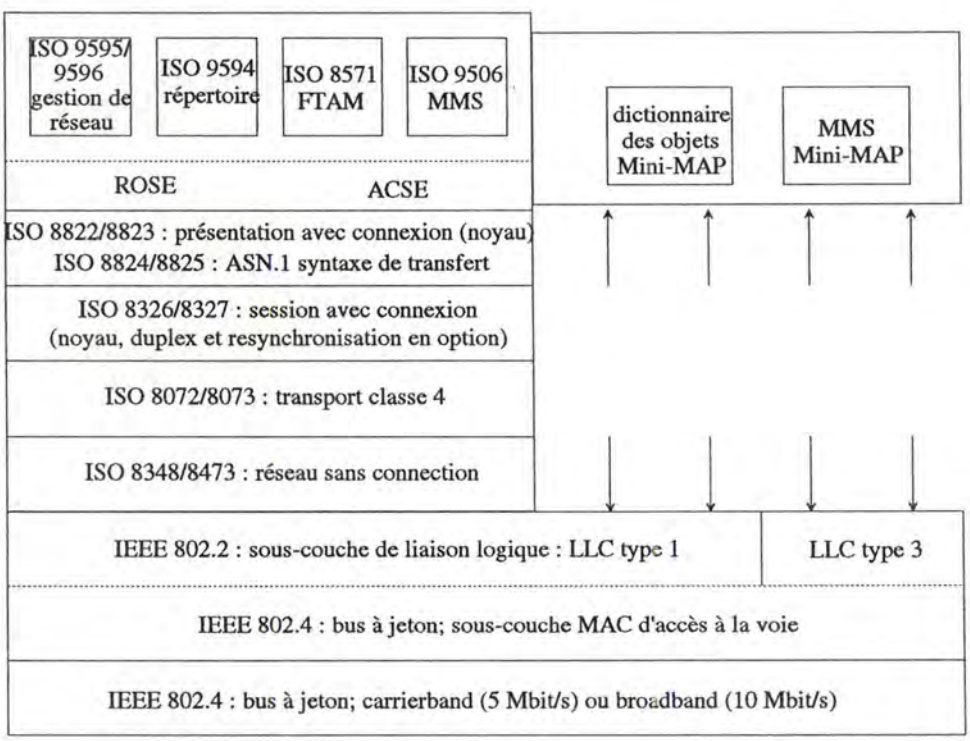


fig. 2.39. L'architecture EPA

L'idée de l'architecture EPA est d'offrir dans une station la pile des protocoles MAP complète et une pile de protocoles OSI qui ne reprenne que les couches 1,2 et 7. Cette pile de protocoles à trois couches serait utilisée pour la transmission rapide de messages, alors que la pile complète permet de communiquer avec n'importe quel équipement MAP.

Cette solution résout le problème de contraintes de temps, mais le développement et la production d'une station MAP/EPA sont encore plus coûteux que ceux d'une station MAP complète puisqu'il faut implémenter dans une même station deux piles de protocoles différentes.

Le Mini-MAP constitue alors la solution à ces problèmes. Les stations Mini-MAP n'ont elles qu'une seule pile de protocoles avec 3 couches : les couches 1, 2 et 7 (*fig. 2.40. L'architecture Mini-MAP*). Pour que ces stations puissent communiquer avec des stations MAP complètes, les messages passent par les stations MAP/EPA qui jouent le rôle de passerelle avec le réseau MAP. (Les stations MAP/EPA sont le plus souvent des contrôleurs de cellule.) En l'absence de stations MAP/EPA sur le réseau Mini-MAP, ce réseau est considéré comme tout autre réseau différent d'un réseau MAP.

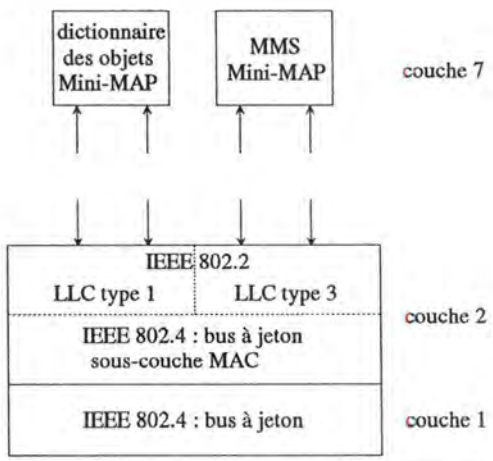


fig. 2.40. L'architecture Mini-MAP

Dans les stations Mini-MAP il est important de faire la distinction entre deux types de stations : celles qui font partie du *token ring* logique c.-à-d. celles qui reçoivent le *token*, et celles qui ne le reçoivent jamais.

Dans le premier cas, on parle de station Mini-MAP active qui possède toutes les fonctions du Mini-MAP. Une station active peut, par le fait qu'elle obtient de temps en temps le *token*, prendre l'initiative d'accéder au bus.

Dans le deuxième cas, celui des stations dites passives, il s'agit de stations toutes simples. Elles ne peuvent pas accéder au bus de leur propre initiative puisqu'elles n'obtiennent jamais le *token* (elles ne font pas partie du *token ring* logique). Une station passive peut seulement communiquer avec une autre station - nécessairement active - qu'après une interrogation de la part d'une station active EPA ou Mini-MAP. Ce processus de *polling* est mis en oeuvre avec le service de réponse immédiate du protocole LLC type 3 et est expliqué plus en détail au chapitre trois concernant le PROFIBUS.

Outre ce service de *polling*, on trouve également un service de *broadcast* qui permet d'envoyer un message à plusieurs stations (actives ou passives) en même temps (sans réponse immédiate évidemment).

En raison de l'absence d'une couche réseau (couche 3), les messages émis par une station Mini-MAP ne transitent que dans le segment Mini-MAP concerné.

La couche application de Mini-MAP offre des services de messagerie MMS avec une structure plus simple et n'utilise pas l'élément de service d'application ACSE. (MMS Mini-MAP comporte donc lui-même les fonctions nécessaires pour gérer une association d'application.) Le deuxième élément de service d'application offert par Mini-MAP est un service simplifié de répertoire (dictionnaire des objets Mini-MAP) qui met en correspondance des noms et des adresses avec une portée limitée au segment Mini-MAP en question.

MMS Mini-MAP offre des catégories de services. D'une part, il y a les services qui se déroulent sur une association explicite. Ces services correspondent à peu près aux services MMS. D'autre part, MMS Mini-MAP offre des services basés sur une communication sans association. Dans ce cas, les données sont envoyées sans faire référence à une association particulière. Ceci introduit naturellement une limitation du fonctionnement et, de ce fait, une réduction des services offerts (*tableau 2.7. Les services MMS Mini-MAP sans association*).

Service MMS Mini-MAP	L'ensemble fonctionnel
Read	Variable Access
Write	Variable Access
Information Report	Variable Access
Status	VMD Management
Identify	VMD Management
Unsolicited Status	VMD Management
Cancel	Environment and General Management
Event Notification	Event Management
Acknowledge Event Notification	Event Management
Input	Operator Communication
Output	Operator Communication

tableau 2.7. Les services MMS Mini-MAP sans association

Pour une station Mini-MAP passive, le nombre de services disponibles se réduit à deux :

- Information Report
- Unsolicited Status

Les grandes différences avec MAP/MMS sont donc :

- L'architecture Mini-MAP ne comprend que 3 couches.
- Il existe des stations Mini-MAP actives et passives.
- Les stations Mini-MAP sont moins complexes et moins coûteuses.
- Sur un réseau Mini-MAP on échange généralement des informations de plus petite taille.
- Pour pouvoir communiquer avec un réseau MAP, il faut nécessairement utiliser une passerelle (*gateway*) qui est souvent une station EPA.
- MMS Mini-MAP ne fait pas appel à l'élément de service d'application ACSE.
- MMS Mini-MAP offre en plus des services basés sur une communication sans association.
- Une station Mini-MAP offre la possibilité de réponse immédiate et de *broadcast*.

2.3. Bref aperçu sur TOP

Parallèlement au développement de MAP chez General Motors, en 1984 est apparu chez Boeing un groupe de travail . Ce groupe de travail avait pour but de définir et d'élaborer un standard de communication de bureau. L'architecture choisie pour le système - appelé TOP (*Technical and Office Protocol*) - est pratiquement la même que celle de MAP. Cette similitude permet une interaction facile entre les deux systèmes pour arriver à un système CIM complet.

La différence entre les deux systèmes réside dans la couche liaison de données et plus précisément dans la sous-couche MAC (*Medium Access Control*). Là où le groupe de travail de GM prévoit l'utilisation d'un *Token Bus*, on définit chez Boeing l'utilisation du réseau IEEE 802.3 (CSMA/CD). Mais Boeing prévoit aussi l'utilisation possible d'un *Token Ring* (IEEE 802.5) ou d'un *Token Bus* (IEEE 802.4). TOP prévoit aussi la possibilité d'envoyer des messages à travers un réseau à paquets (X.25).

A la couche application, de l'architecture de TOP, on retrouve grosso-modo les mêmes éléments de service d'application qu'à la couche application de MAP. Il est évident que la messagerie industrielle (MMS) n'est pas un élément de services d'application de TOP. Mais pour cet usage, d'autres éléments de services d'application tels que le courrier électronique (X.400 '84) et le terminal virtuel (*virtual terminal*) sont offerts par TOP.

La figure 2.41. montre l'architecture de TOP.

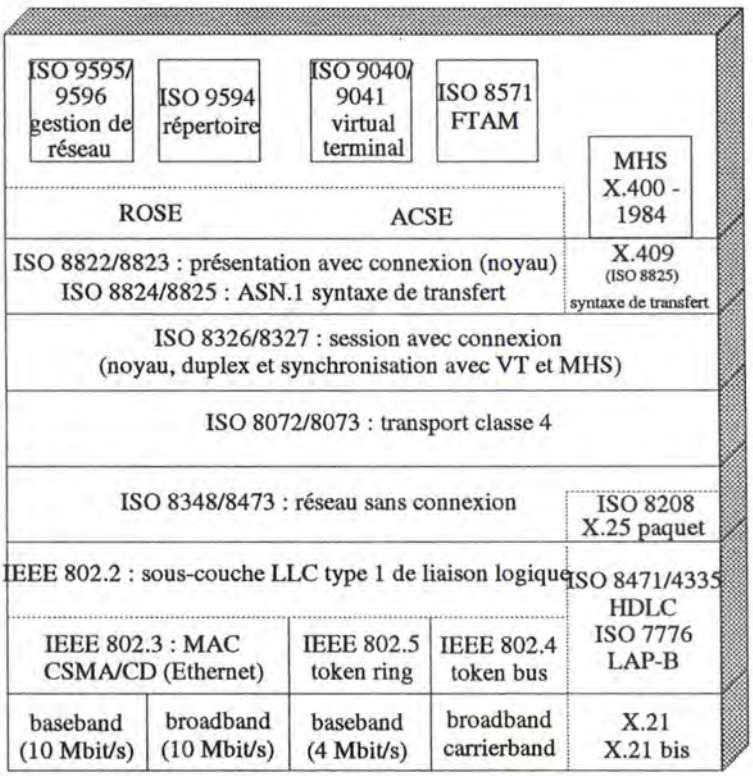


fig. 2.41. L'architecture de TOP

3. Une norme nationale allemande : Le PROFIBUS[17]

Le chapitre 2 nous a introduit dans une norme internationale du domaine de l'automatisation qui est le réseau MAP et nous avons surtout parlé de la messagerie industrielle MMS.

La solution proposée par MAP pour la communication entre équipements de fabrication est bonne pour le réseau d'usine, mais elle ne répond pas vraiment aux besoins d'un réseau de terrain.

Un premier pas dans la bonne direction fut la normalisation de MAP/EPA et des stations Mini-MAP qui tiennent déjà compte des contraintes de temps de réponse et de taille des données exigées pour un réseau de terrain. Mais l'architecture Mini-MAP est en elle même assez controversée (réf. /13/).

Le manque d'une solution adéquate, le souhait et la nécessité de pouvoir communiquer facilement via un réseau avec les équipements des niveaux 1 et 2 ont amené les constructeurs à mettre au point des solutions. C'est ainsi que des groupes de travail se sont formés dans les différents pays. Des solutions différentes ont été développées. En Allemagne, on parle du PROFIBUS (DIN 19245 1/2 - PROcess FIeld BUS), en France, on parle du FIP (*Factory Instrumentation Protocol*) et, aux Etats-Unis, on parle du Bitbus. A côté des efforts nationaux, il existe aussi un groupe de travail qui s'occupe de l'élaboration d'une norme internationale de réseau de terrain. Il s'agit du projet ISA SP50. Cette norme internationale n'est pas attendue avant 1995 alors que le PROFIBUS[18] est disponible dès maintenant.

C'est aussi du PROFIBUS dont il sera question dans ce chapitre parce que c'est la solution la plus avancée en ce moment (on ne dit pas pour autant que c'est la meilleure). Mais avant de présenter le PROFIBUS, essayons d'abord de dire ce qu'est un réseau de terrain et quelles exigences on pose à ce type de réseau.

Le réseau de terrain a pour but de relier des équipements du niveau 1 (niveau de terrain) entre eux et avec des équipements du niveau 2 (niveau de station). Un réseau de terrain peut également être utilisé pour relier des équipements de niveau 2 entre eux. (cfr. *fig. 1.4. Les standards dans la communication hiérarchique*)

17 Nous nous sommes basés principalement sur les références /1/, /4/, /5/ et /23/.

18 Ici on fait un abus de langage en parlant du PROFIBUS comme quelque chose de physique alors qu'il ne s'agit que d'une norme. Cette formulation sera utilisée à plusieurs reprises dans la suite.

Le réseau de terrain (*fieldbus*) fait partie des réseaux locaux (LAN) connus sous le nom de *Low-Cost*-LAN (réseau local à coûts bas) utilisé pour relier les niveaux 1 et 2. Ce type de réseau répond à différentes contraintes :

- ❒ le réseau de terrain doit être adapté à l'environnement industriel
- ❒ le réseau de terrain doit être capable de travailler en temps réel :
  - le temps de réaction est très court,
  - les messages envoyés sont de petite taille,
  - l'accès au bus doit être déterministe,
  - le protocole utilisé doit être assez simple (pour ne pas avoir un *overhead* trop large et pour ne pas perdre trop de temps).
- ❒ le protocole doit offrir plusieurs priorités.
- ❒ le réseau de terrain doit offrir différentes vitesses de transmission .
- ❒ le réseau de terrain doit être fiable (on exige par exemple une distance de hamming - *hamming distance* - égale à 4).
- ❒ la connexion au bus doit être bon marché.
- ❒ le câble utilisé pour réaliser le bus doit être simple et bon marché. On utilise souvent de la paire torsadée (*twisted pair*) ou un câble coaxial simple.
- ❒ le réseau de terrain devrait être compatible à MAP pour permettre une connexion facile à un réseau MAP.

3.1. Présentation du PROFIBUS

Pour présenter le PROFIBUS, nous ferons d'abord la relation avec l'ISORM et nous montrerons ensuite l'architecture du PROFIBUS. Enfin, nous expliquerons en détail les différentes couches du PROFIBUS. Nous donnerons pour chaque couche les éléments fondamentaux, mais aussi les principes et les concepts de la couche.

Relation avec l'ISORM

Le groupe[19] qui a élaboré et défini la norme Profibus s'est fortement inspiré de l'ISORM. On peut, par conséquent, faire immédiatement la relation entre l'architecture du Profibus et celle de l'ISORM (*fig. 3.1. Transposition de l'architecture du Profibus sur l'ISORM*).

19 Ce groupe est formé par des fabriquants tels que AEG, Honeywell, Phoenix, Siemens et par des Instituts et des Universités.

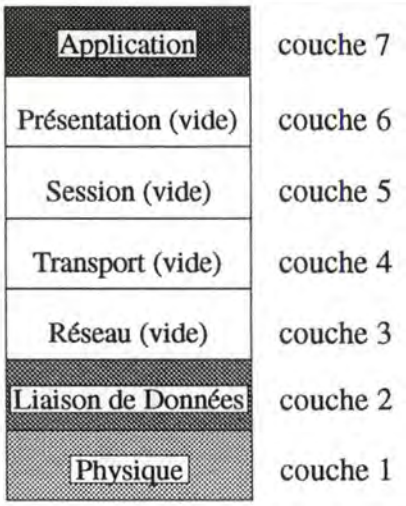


fig. 3.1. Transposition de l'architecture du Profibus sur l'ISORM

L'architecture du Profibus se base sur 3 couches :

- la couche application (couche 7)
- la couche liaison de données (couche 2)
- la couche physique (couche 1)

La couche physique se base sur une interface RS 485 et transmet les données par paire torsadée (*twisted pair*).

La couche 2, appelée FDL (*Fieldbus Data Link*), est composée de deux sous-couches :

- la sous-couche FLC (*Fieldbus Link Control*)
- la sous-couche MAC d'accès à la voie

(*fig. 3.2. La couche FDL*)

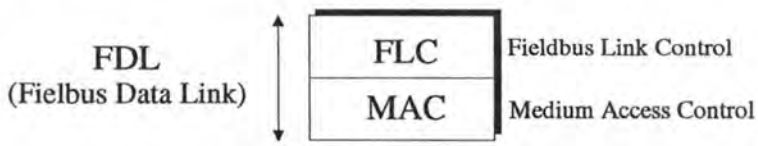


fig. 3.2. La couche FDL

Dans le cas du Profibus, la couche 7 est également formée de plusieurs sous-couches :

- la sous-couche FMS (*Fieldbus Message Specification*)
- la sous-couche LLI (*Lower Layer Interface*)

(*fig. 3.3. La couche 7 du Profibus*)

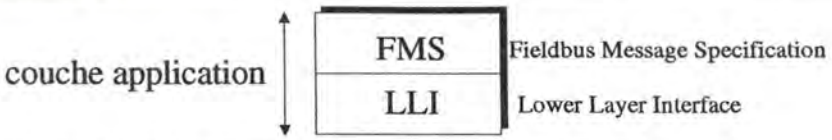


fig. 3.3. La couche 7 du Profibus

A côté des trois couches du Profibus, on trouve encore une quatrième qui ne fait pas directement partie de l'architecture Profibus. Il s'agit de l'interface entre le processus d'application et la couche 7. Cette interface est l'ALI (*Application Layer Interface*).

La nécessité et la fonctionnalité des différentes couches et sous-couches sont expliquées au paragraphe suivant, où on trouvera aussi les principes et les concepts fondamentaux du Profibus.

Les principes et les concepts fondamentaux

L'architecture du Profibus se divise, comme nous l'avons déjà vu, en deux grandes parties. D'une part, on a les couches inférieures (les couches 1 et 2) définies dans DIN 19245/1 où on parlera surtout de la couche FDL. D'autre part, on a la couche application (couche 7) définie dans DIN 19245/2.

Dans l'étude du PROFIBUS, nous respecterons cette division en deux parties. Puisque les concepts sont fortement liés à une partie, ils sont expliqués lors de la présentation des couches qui fait l'objet des pages suivantes.

① La couche FDL (Fieldbus Data Link)

La couche FDL est composée de 2 sous-couches : la sous-couche FLC (*Fieldbus Link Control*) et la sous-couche MAC. A côté du contrôle d'accès au bus et de la gestion du *token*, la tâche de la FDL consiste aussi à offrir des services de transmission de données à l'utilisateur de la couche FDL (le *FDL user*).

Nous présenterons d'abord la sous-couche MAC. Ensuite, nous donnerons une explication sur les différents services offerts par la couche FDL.

Bien que la gestion du *token* soit quelque chose de fondamental dans la philosophie du Profibus, nous n'en dirons que quelques mots.[20]

20 Plus de détails sur la gestion du token dans le Profibus se trouvent dans les références /1/ et /4/.

La sous-couche MAC (*Medium Access Control*)

La sous-couche MAC doit accomplir une double fonctionnalité pour les réseaux de terrain. Premièrement, elle doit assurer que, dans la communication entre appareils qui ont une certaine intelligence, chaque appareil reçoit pendant une période définie une tranche de temps suffisante pour échanger de l'information. La sous-couche MAC doit aussi garantir que chacun de ces appareils a la possibilité d'envoyer des messages pendant la période déterminée.

En second lieu, la sous-couche MAC doit permettre l'échange d'informations de haut niveau en temps réel entre un appareil de niveau 2 et un appareil de niveau 1.

On présentera brièvement les méthodes (principes) optimisées qui existent pour ces deux tâches de la sous-couche MAC.

La première méthode est le principe du maître/esclave (*master/slave*) ou de *polling*. Dans ce principe, une station, le maître, peut accorder un droit d'émission (limité dans le temps) à un esclave (*fig. 3.4. Principe du maître/esclave*).

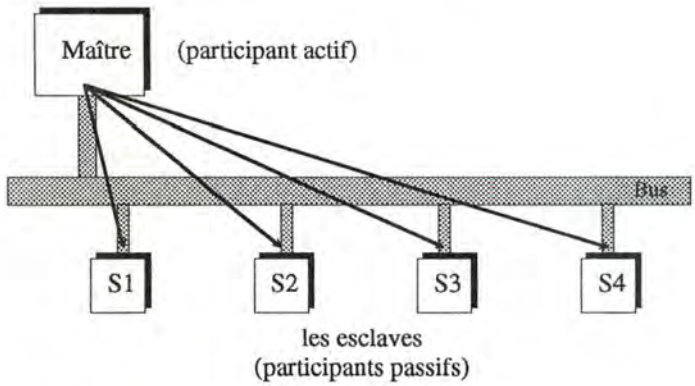


fig. 3.4. Principe du maître/esclave

La deuxième méthode est le principe du *token passing* qui permet de répartir l'accès aux bus entre des stations équitables (*gleichberechtigt*). Dans cette méthode, une suite de bits (le *token*) parcourt les stations connectées. La station qui détient le *token* peut émettre. Le temps pendant lequel le *token* reste dans une station est déterminé. Le temps de rotation (TRT-*Token Rotation Time*) est défini/fixé au départ.

Comme le parcours du *token* forme un cercle logique (*ring*) on parlera dans la suite aussi de *token ring* logique lorsqu'on parle de *token* bus. La figure 3.5. montre le schéma d'un *token* bus (ou *token ring* logique). Nous avons déjà donné une

représentation du token bus à la figure 2.6., mais ici nous avons choisi une autre représentation.

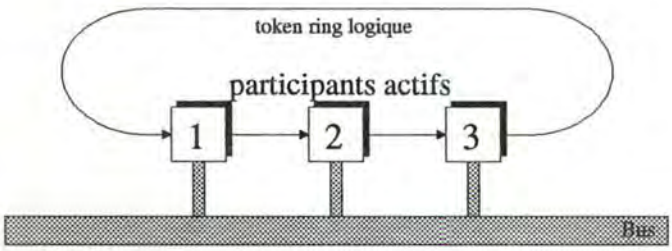


fig. 3.5. Le token bus ou le token ring logique

La gestion du *token* est assez complexe et doit en outre être implémentée dans chaque station. Mais l'avantage majeur de cette méthode c'est que l'accès au bus est déterministe ce qui est crucial dans l'environnement industriel.

Comment ces deux méthodes vont-elles s'articuler dans le concept du Profibus?

Beaucoup de systèmes d'automation de bas niveau fonctionnent selon le principe du maître/esclave (*master/slave*) où on a une station maître et plusieurs stations esclaves. Le maître communique généralement de manière cyclique avec chaque station esclave.

Mais d'autre part, on souhaite aussi relier des équipements de niveau 2 (et de niveau 1) entre eux. Un exemple typique est donné par un PG* ou un OS* local. Pour ce système, la méthode du *token passing* offre de nombreux avantages pour la communication entre équipements complexes. L'avantage majeur est que l'accès au bus est déterministe.

Profibus offre un accès déterministe au bus. Des participants actifs[21] accèdent au bus par le principe du *token passing* alors que la méthode sous-jacente

21 **Les participants actifs** sont des stations qui participent activement au token ring logique. Il s'agit de stations intelligentes qui peuvent accéder au bus dès qu'elles ont le token.

(*underlaying/unterliegend*) du maître/esclave[22] est utilisée pour la communication entre les stations actives et les stations passives[23].

Cet accès hybride répond aux exigences de performance, de temps de réponse, de fiabilité et de complexité. La figure 3.6. donne une structure typique avec 3 participants actifs et 4 participants passifs. Le *token* circule entre les participants actifs selon le chemin d'un *token ring* logique.

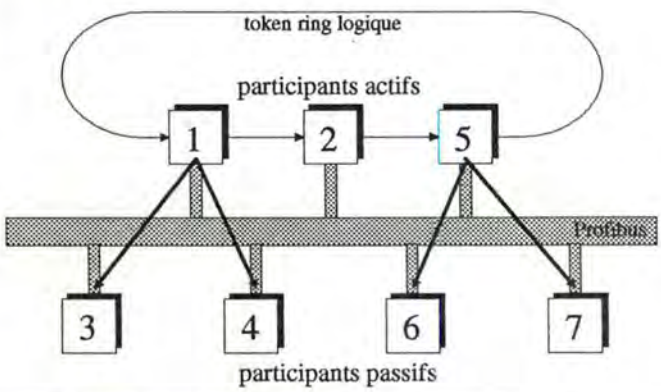


fig. 3.6. Structure typique de l'accès dans Profibus

Services offerts par la couche FDL au *FDL-User* (couche 7)

La couche FDL offre trois services non cycliques et un service cyclique au FDL-*User* :

1. **SDN** : *Send Data with No acknowledge*
   (envoyer sans acquittement)
2. **SDA** : *Send Data with Acknowledge*
   (envoyer avec acquittement)
3. **SRD** : *Send and Request Data with reply*
   (envoyer avec demande de données et des données de retour)
4. **CSRD** : *Cyclic Send and Request Data with reply*
   (SRD cyclique)

22 Profibus définit un cas particulier qui est celui d'esclave avec initiative. Il s'agit d'un esclave pur et simple qui ne contient pas toute la gestion du token, mais qui, lorsqu'on lui adresse la parole peut prendre l'initiative de lancer un service (non confirmé) de sa propre initiative. Ce mécanisme est utilisé pour permettre à une structure assez simple d'envoyer des messages d'urgence (ex. des messages d'alarmes à un maître autre que celui qui lui a adressé la parole). En l'absence de ce type, il aurait fallu utiliser un maître avec toute sa complexité de gestion de token.

23 Les **stations passives** sont des stations qui ne participent pas au token ring logique. Ces stations ne reçoivent pas le token et ne peuvent accéder au bus que lorsqu'une station leur demande de le faire.

Ces quatre services sont optionnels et ne doivent en conséquence pas nécessairement être tous implémentés dans une station particulière. Mais il va de soi qu'au moins un des quatre services doit y être présent.

Pour réaliser ces services, on donne des primitives de service accessibles au FDL-User. Nous allons présenter brièvement les quatre services et les primitives nécessaires pour leur réalisation.

SDN : Le service SDN permet à l'utilisateur de la couche FDL d'une station active (nommée dans la suite *local-user*) d'envoyer des données (des LSDU) a une, plusieurs ou à toutes les autres stations en même temps. Le *local-user* reçoit une confirmation interne de l'envoi des données, mais ne reçoit aucune confirmation de la bonne ou mauvaise réception des donnée chez le *remote-user* (l'utilisateur de la couche FDL de la station répondeur).

Le tableau 3.1. reprend les primitives de service offertes pour réaliser le service SDN et indique chaque fois le type de participant nécessaire. La lettre A indique une station active, la lettre P une station passive. La figure 3.7. montre l'enchaînement des primitives sur un diagramme séquence-temps.[24]

FDL_Data.req	A
FDL_Data.ind	A/P
FDL_Data.con	A

tableau 3.1. Primitives de service du service SDN

Le service SDN est utilisé pour envoyer des messages *broadcast* ou *multicast.*

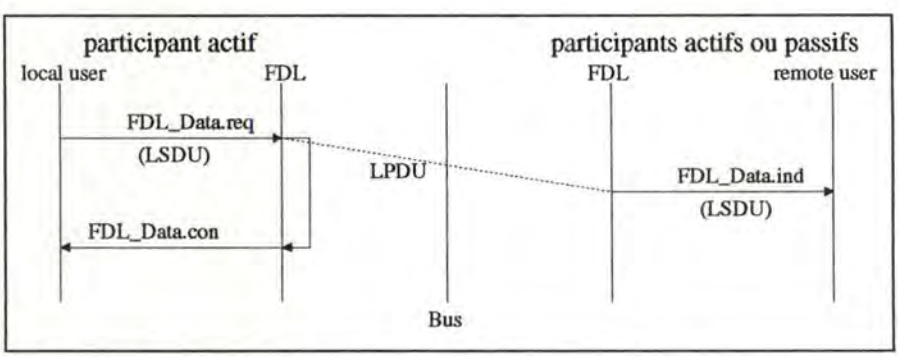


fig. 3.7. L'enchaînement des primitives de service du service SDN

24 .req ⇒ .request
.ind ⇒ .indication
.con ⇒ .confirmation

SDA : Le service SDA permet au *local-user* d'envoyer des LSDU à un seul *remote-user*. Le remote-user va informer le *local user* de la bonne ou de la mauvaise réception. En cas d'erreur de transmission, la couche FDL du *local user* retransmet les données.

Le tableau 3.2. reprend les primitives de service offertes pour réaliser le service SDA et indique chaque fois le type de participant nécessaire. La lettre A indique une station active, la lettre P une station passive. La figure 3.8. montre l'enchaînement des primitives sur un diagramme séquence-temps.

FDL_Data_Ack.req	A
FDL_Data_Ack.ind	A/P
FDL_Data_Ack.con	A

tableau 3.2. Primitives de service du service SDA

Le service SDA permet aux stations actives d'accéder au bus.

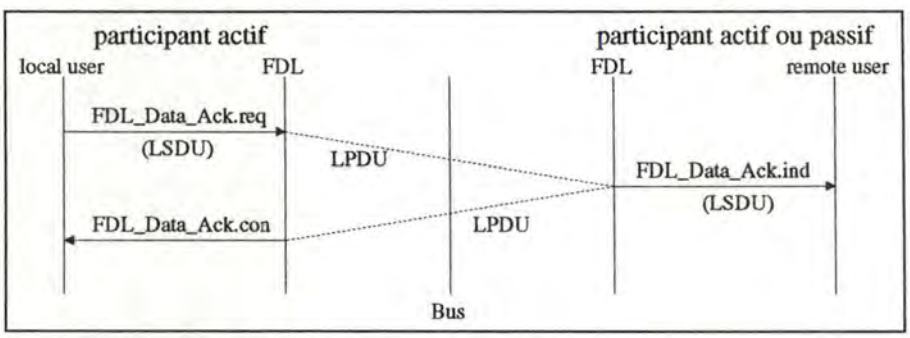


fig. 3.8. L'enchaînement des primitives de service du service SDA

SRD : Le service SRD permet au *local-user* d'envoyer des données à un seul *remote-user* et de recevoir en même temps des données du remote-user que celui-ci a mis à la disposition du *local-user*. En cas d'erreur de transmission, la couche FDL réémet les données.

Le tableau 3.3. reprend les primitives de service offertes pour réaliser le service SRD et indique chaque fois le type de participant nécessaire. La lettre A indique une station active, la lettre P une station passive. La figure 3.9. montre l'enchaînement des primitives sur un diagramme séquence-temps.

FDL_Data_Reply.req	A
FDL_Data_Reply.ind	A/P
FDL_Data_Reply.con	A
FDL_Reply_Update.req	A/P
FDL_Reply_Update.con	A/P

tableau 3.3. Primitives de service du service SRD

Le service SRD est couramment utilisé lorsqu'on rencontre des stations actives et des stations passives sur le même réseau de terrain.

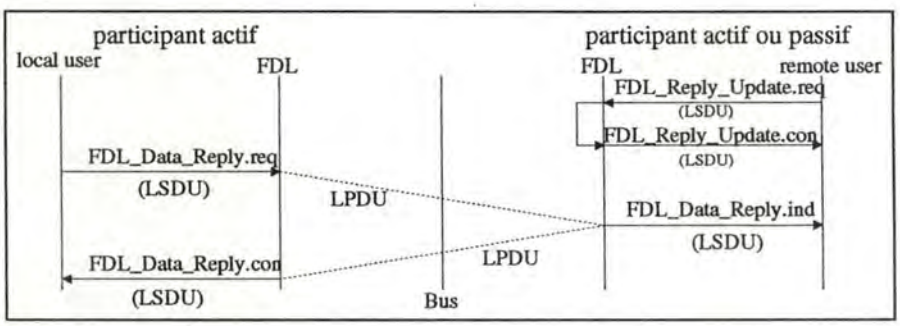


fig. 3.9. L'enchaînement des primitives de service du service SRD

CSRD : Le service CSRD permet au *local-user* d'envoyer et de recevoir des données de plusieurs *remote-user* de manière cyclique. Il s'agit donc d'un service SRD appliqué cycliquement à chaque station d'un ensemble de stations bien définis. En cas d'erreur de transmission, la couche FDL réémet les données.

Le tableau 3.4. reprend les primitives de service offertes pour réaliser le service CSRD et indique chaque fois le type de participant nécessaire. La lettre A indique une station active, la lettre P une station passive. La figure 3.10. montre l'enchaînement des primitives sur un diagramme séquence-temps.

FDL_Send_Update.req	A
FDL_Send_Update.con	A
FDL_Cyc_Data_Reply.req	A
FDL_Cyc_Data_Reply.con	A
FDL_Cyc_Entry.req	A
FDL_Cyc_Entry.con	A
FDL_Cyc_Deact.req	A
FDL_Cyc_Deact.con	A
FDL_Data_Reply.ind	A/P
FDL_Reply_Update.req	A/P
FDL_Reply_Update.con	A/P

tableau 3.4. Primitives de service du service CSRD

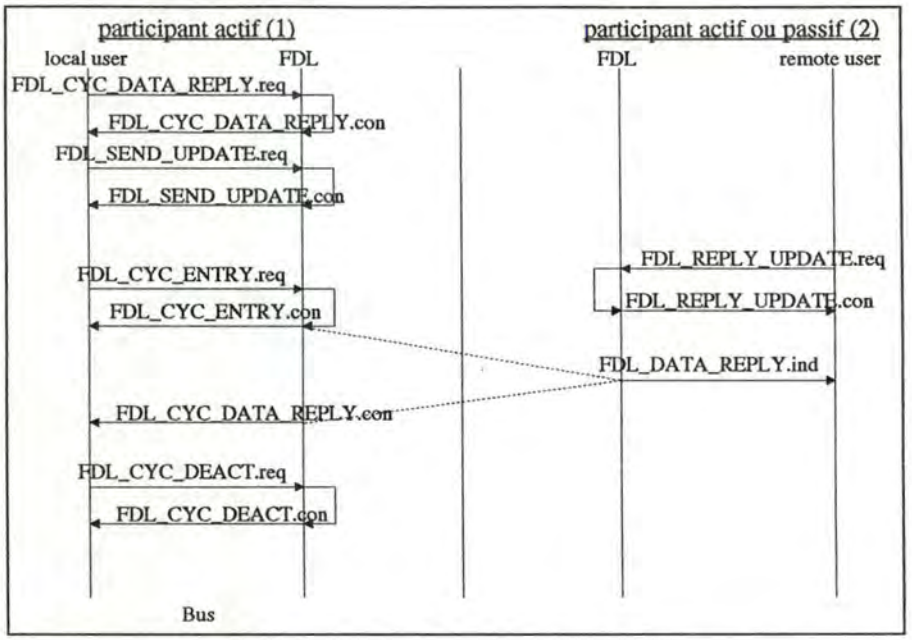


fig. 3.10. L'enchaînement des primitives de service du service CSRD

La gestion du *token*

La gestion du *token* est importante, mais ne concerne, en principe, que les participants actifs.

Chaque participant actif se constitue au départ une liste des stations actives (LAS = *List of Active Stations*) et passe le *token* à la station qui le suit dans la liste, le dernier participant le passe au premier. La gestion du *token* permet aussi d'ajouter dynamiquement des stations qui sont reconnues automatiquement et qu'on peut aussi retirer. En outre, la gestion du *token* s'occupe du contrôle du temps de rotation, de la duplication du *token* et de la perte d'un *token*.

La différenciation entre haute priorité et basse priorité se fait aussi à cet endroit. Une station peut ainsi toujours envoyer un message à haute priorité même si le temps de rotation du *token* est déjà dépassé.

Pour plus de détails sur la gestion du *token* nous conseillons de consulter les références /4/ et /1/.

Pour terminer cette présentation de la couche FDL, nous allons dire un mot, sans entrer dans les détails, de la sous-couche FLC. Cette couche peut être représentée par l'ensemble des états nécessaires pour un déroulement correct de la

communication. On y retrouve notamment des états tels que *offline, listen token, active idle, passive idle, use token, claim token, ...*

Nous n'expliquons pas les différents états et nous ne donnons pas le diagramme états-transitions puisqu'ils sont bien expliqués dans DIN 19245/1 (ref. /4/).

② La couche Application (*Application Layer*)

La couche application du Profibus est elle aussi divisée en plusieurs sous-couches. Une première sous-couche est le FMS (*Fieldbus Message Specification*), un *subset* de MMS. Une deuxième sous-couche est le LLI (*Lower Layer Interface*) qui joue le rôle d'interface entre la couche FDL et la sous-couche FMS.

Nous parlons aussi d'une troisième «sous-couche» qui est le l'ALI (*Application Layer Interface*). L'ALI joue l'interface entre le processus d'application (*application process*) proprement dit et la couche application. (*fig. 3.12. La couche application du Profibus*)

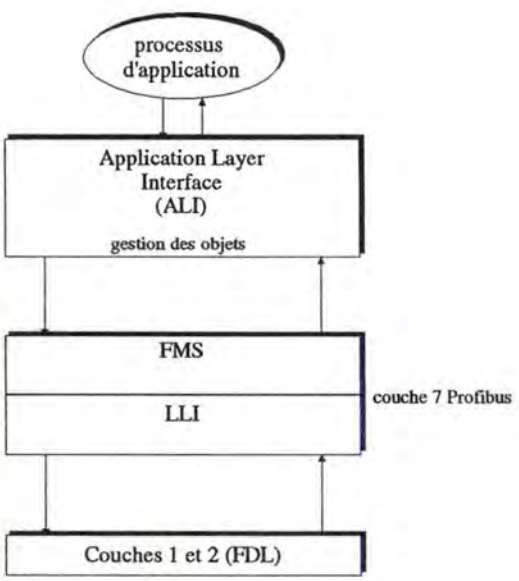


fig. 3.12. La couche application du Profibus

Avant de présenter plus en détail les sous-couches FMS et LLI, parlons d'abord des concepts utilisés dans la couche application. Nous parlons du modèle client/serveur, du concept de VFD (*Virtual Field Device*), du modèle d'objet et de la KBL (*Kommunikationsbeziehungsliste*). A ceci, s'ajoute le concept d'ALI qui est

crucial pour le bon fonctionnement d'une application utilisant le Profibus. Par conséquent nous commençons par l'ALI.

ALI - *Application Layer Interface*

L'ALI joue le rôle d'interface entre un processus d'application et la couche application. Cette interface est nécessaire puisque dans les systèmes ouverts, les normes ne concernent que la communication. Le sens réel des données échangées ne dépend que de l'application.

Dans la communication entre deux processus d'application, il est nécessaire de rendre, par des services, les objets locaux accessibles. Les services sont des opérations sur des objets d'une classe définie.

Le but de l'ALI est donc de faire la transposition des objets locaux existants (*Prozessobjekte*) sur des objets connus par le PROFIBUS (*Kommunikationsobjekte*) et d'assurer le transfert des services aux couches de communication (couches 7, 2, 1).

La partie du processus d'application visible pour la communication est modélisée par le VFD (*Virtual Field Device*) qui fournit une vue virtuelle standardisée de l'équipement. Pour pouvoir fonctionner, l'ALI utilise aussi des directory d'objets (*Objektverzeichnisse*).

L'ALI peut donc être d'une part le client et d'autre part le serveur (puisqu'il contient le VFD) dans une relation client/serveur.

Le modèle client/serveur (*client/server model*)

Le concept du modèle client/serveur du Profibus est le même que celui expliqué au paragraphe 2.2.2., paragraphe qui parle de la messagerie industrielle MMS.

Pour les services FMS, nous partirons généralement de ce modèle qu'il s'agisse d'une communication maître-maître (*master-master*) ou maître-esclave (*master-slave*).

La différence avec les services asymétriques purs d'OSI réside dans le fait que dans le Profibus, il est pensable que deux stations changent leur rôle. Ceci peut être le cas si deux maîtres communiquent par une relation de communication acyclique. Ainsi, chaque station peut, à tout moment, être l'initiateur de services (client) et le répondeur à des services (serveur). (réf. /23/)

Le modèle de VFD (*Virtual Field Device*)

Un équipement virtuel est en général une abstraction d'une classe d'équipements réels offerts à un utilisateur (*user*).[25] Chaque classe d'équipements est caractérisée par des attributs. Ces attributs sont représentés dans Profibus par la partie du processus d'application manipulable par des services du Profibus. Cette partie du processus d'application est composée d'objets de processus (*Prozessobjekte*) qui sont déclarés objets de communication (*Kommunikationsobjekte*).

Il est possible que plusieurs VFD se situent dans une même station. Dans un cas de ce genre, chaque VFD peut être adressé séparément.

Le modèle d'objet

A la base de toute communication dans Profibus se situent des objets qui permettent l'échange de données structurées. Les objets sont caractérisés par des attributs et par des opérations possibles et définies pour chaque classe d'objets.

Dans Profibus, on fait, d'une part, la distinction entre objets statiques et objets dynamiques. La différence entre les deux réside dans le moment de la définition de l'objet. Les objets statiques sont des variables (*simple*, *array* et *record*), des domaines et des événements et ne peuvent pas être détruits. Les objets dynamiques sont des listes de variables (*variable-list*) et des instances de programme (*program invocation*) et peuvent être créés et détruits lors de l'exécution.

On fait d'autre part la différence entre objets explicites et objets implicites. Les objets explicites (les variables, par exemple) doivent être décrits explicitement dans le directory d'objets (*Objektverzeichnis* - OV) avant de l'utilisation, alors que les objets implicites (l'objet VFD, par exemple) sont décrits implicitement.

L'accès aux objets peut se faire de quatre manières d'adressage différentes. Premièrement, on peut parler d'un adressage logique réalisé par des adresses très courtes (nommées "indices" dans Profibus). Les indices sont référencés dans l'OV de l'équipement de terrain.

25 Des équipements virtuels sont, par exemple, la mémoire (*virtual memory*), les écrans (*virtual terminal*) ou même les équipements de terrain (*Feldgeräte*).

Deuxièmement, on a la possibilité d'accéder à un objet via son adresse physique, ce qui correspond à l'adressage physique.

Troisièmement, l'adressage implicite est utilisé pour adresser le VFD et le processus d'application.

Quatrièmement, l'adressage par nom permet de référencer l'objet par son nom au lieu de le faire par son adresse logique. Le nom (longueur maximale : 32 caractères) est souvent prédéfini par les profiles[26].

Un élément important du modèle d'objet dans Profibus est l'existence d'un dictionnaire d'objet (OV). L'entrée d'un objet de processus (*Prozessobjekt*) dans cet OV en fait un objet de communication (*Kommunikationsobjekt*).

Chaque station définit ses objets dans un dictionnaire d'objets (*source*-OV) mais elle a aussi un OV de toute la description ou d'une partie de la description des objets des stations paires (*remote*-OV). Pour manipuler le *source*-OV via le bus, Profibus propose des services de management.

Chaque OV peut avoir jusqu'à 6 sous-dictionnaires, dont chacun contient la description d'une classe d'objets de communication.

La KBL (*Kommunikationsbeziehungsliste*)

Dans Profibus, différentes relations de communication sont possibles :

1. des relations de communication sans connexion
   - Broadcast (toutes les stations connectées)
   - Multicast (plusieurs stations connectées)
2. des relations de communication avec connexion
   - maître-maître
   - maître-esclave

Ce qui fait l'originalité par rapport à d'autres systèmes, c'est que chaque relation de communication est préprojetée (prédéfinie et décrite). Ceci est nécessaire pour répondre aux exigences de réponses en temps réel que pose un réseau de terrain.

Lors du lancement d'une station, ces relations de communication sont définies dans une liste des relations de communication (*Kommunikationsbeziehungsliste* - KBL).

26 Les profiles sont comparables aux *companion standards* de MMS.

Le système de KBL a été possible pour les relations de communication des équipements d'un réseau de terrain parce que pour ces équipements, on sait d'avance avec quel autre équipement ils vont communiquer. [En général, pour des équipements autres que ceux connectés à un réseau de terrain, on ne sait pas prévoir toutes les connexions qu'on peut avoir. Donc, une préprojection des relations de communication ne serait dans ce cas que partiellement possible.]

La sous-couche FMS (*Fieldbus Message Specification*)

La sous-couche FMS offre au FMS-user des services comparables aux services MMS[27]. Ces services ne sont pas nécessairement disponibles pour tous les équipements, ce qui nous paraît évident.

Les services offerts par FMS répondent entre autres aux critères suivants :

1. L'ensemble des services offerts est adapté aux besoins des réseaux de terrain.
2. Il y a moyen de se connecter facilement à un réseau MAP/MMS.
3. Les critères stricts de temps de réponse doivent être respectés.

La sous-couche FMS réalise aussi la transformation des services FMS en des services LLI et rend les spécificités du réseau de terrain (ex. communication cyclique ou non cyclique) invisibles pour l'utilisateur.

27 En fait, FMS est un *subset* de MMS.

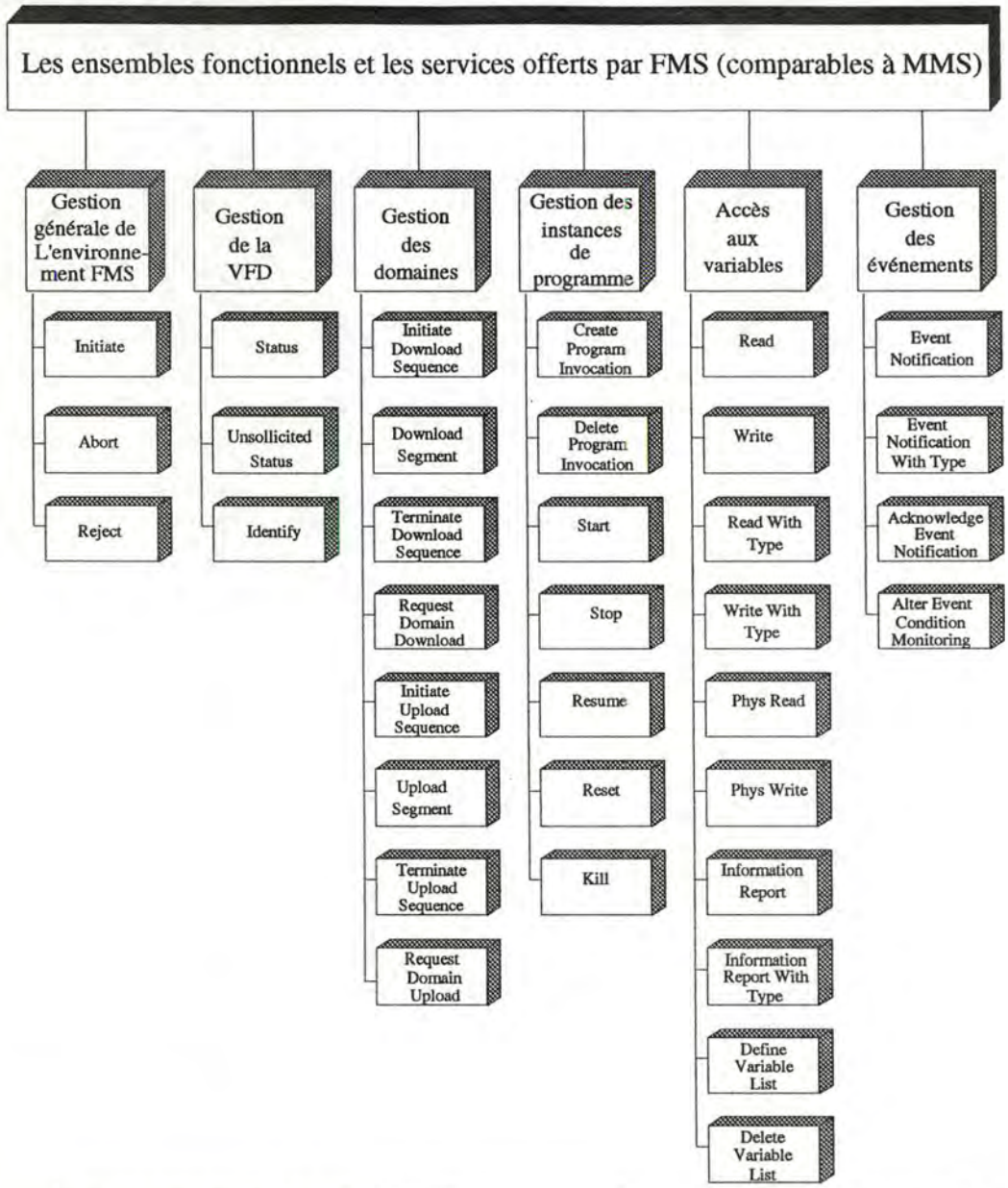


fig. 3.13. Les services FMS comparables aux services MMS

Les figures 3.13. et 3.14. montrent respectivement les services FMS comparables aux services MMS et les services FMS différents de ceux de MMS. Le groupe Profibus s'est fortement inspiré de MMS dans la définition des services et dans leur répartition dans des ensembles fonctionnels comparables à ceux présentés aux chapitre 2.

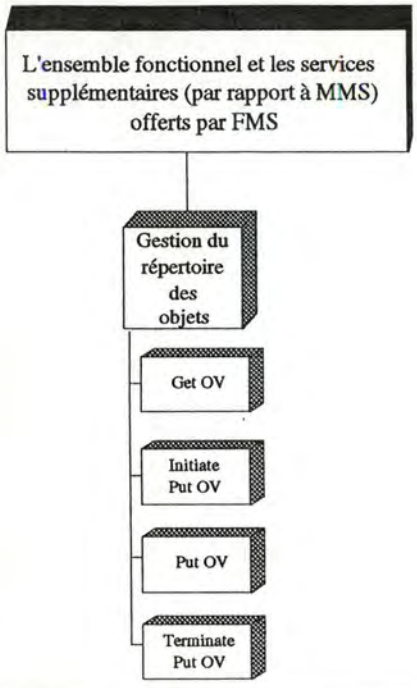


fig. 3.14. Les services FMS autres que ceux de MMS

La sous-couche LLI (*Lower Layer Interface*)

La tâche principale de la sous-couche LLI est de réaliser la transposition des fonctionnalités spécifiques des réseaux de terrain à des services FMS et inversément. Ces fonctionnalités spécifiques sont :

- la communication maître-maître
- le polling cyclique dans une communication maître-esclave
- la réponse immédiate (*immediate response*)
- le broadcast et le multicast

A côté de cela, le LLI s'occupe du contrôle des flux et remplace donc en partie la couche 4 du modèle OSI. Le LLI réalise aussi l'ouverture et la fermeture de la connexion en substituant ainsi l'ACSE.

Plus globalement, on peut dire que le LLI remplace en partie les couches 3 à 6 du ISORM.

Nous terminons ici notre brève présentation du Profibus. Nous ne devons jamais oublier qu'on sait qui va communiquer avec qui, dans le cas des équipements

connectés à un réseau de terrain. La préprojection des relations de communication dans la KBL est une des spécificités du Profibus.

Profibus définit aussi des services de management, mais nous n'en parlerons pas ici. Ceux qui sont intéressés peuvent toujours consulter DIN 19245 1 et 2 (références /4/ et /5/).

Pour terminer ce chapitre sur le Profibus, nous allons établir une comparaison entre le Profibus et les normes internationales vues au chapitre 2.

3.2. Comparaison de PROFIBUS avec MAP/MMS et Mini-MAP

Jusqu'à maintenant, nous avons vu l'existence de trois standards : MAP complet, EPA et les stations Mini-MAP et le Profibus.

MAP est souvent utilisé comme épine dorsale. De plus, l'architecture du MAP complet est basée sur l'ensemble des 7 couches OSI. Pour ces raisons, nous ne comparerons pas MAP et Profibus. La seule chose intéressante à remarquer est que FMS de Profibus est un subset de MAP/MMS, ce qui permet un passage aisé de l'un vers l'autre.

Nous pouvons aussi constater qu'une comparaison entre EPA et Profibus est peu utile puisque les stations EPA constituent en fait des *gateway* entre un réseau Mini-MAP et un réseau MAP complet. Par ce fait, les stations EPA intègrent deux piles de protocoles (MAP complet et Mini-MAP) ce qui n'est pas du tout comparable à Profibus.

Comparons des choses comparables et Mini-MAP au Profibus.

L'architecture du Profibus tout comme celle de Mini-MAP est basée sur 3 couches qui correspondent aux couches 7, 2 et 1 lorsqu'on fait la transposition sur le modèle OSI. Mais ce n'est pas la seule ressemblance entre les deux réseaux.

En fait, dans les deux cas on distingue station active et station passive. Les deux réseaux offrent la possibilité de réponse immédiate et l'envoi de messages broadcast.

Aucun des deux n'utilise l'élément de service d'application ACSE.

Par contre, alors qu'ils utilisent tous deux la même méthode d'accès pour les stations actives (le *token passing*) et pour les stations passives, le Profibus offre un type de station supplémentaire parfaitement adapté à l'environnement industriel : celle d'une station passive avec initiative. Dès qu'on envoie un message à une station passive avec initiative, celle-ci peut prendre l'initiative d'accéder au bus pour envoyer un message d'urgence à une station active quelconque.

Pour Mini-MAP et pour Profibus on a dû remplacer certaines fonctionnalités des couches 3 à 6 du modèle OSI. Dans Mini-MAP, on ne trouve presqu'aucune fonctionnalité de ces quatre couches. De ce fait, son architecture est très controversée. Pour le Profibus, c'est surtout la sous-couche LLI qui reprend des fonctionnalités des couches absentes.

Une originalité du Profibus est l'existence de la KBL. Toutes les relations de communication sont préprojetées et reprises dans cette KBL. Cette liste permet aussi d'atteindre l'objectif de la possibilité de réponses en temps réel. Elle offre donc un avantage non négligeable pour les réseaux de terrain.

Ce qu'il faut encore signaler c'est la différence entre les deux dictionnaires d'objets et la différence entre l'API (*Application Programming Interface*) et l'ALI (*Application Layer Interface*).

Mini-MAP et Profibus ont tous les deux un dictionnaire d'objets. Une première différence c'est que pour Mini-MAP ce dictionnaire se situe au niveau de la couche 7 tandis que pour le Profibus il se situe dans l'ALI. La deuxième différence consiste dans le contenu des dictionnaires d'objets. Le dictionnaire de Mini-MAP contient les noms et les attributs des objets alors que celui du Profibus contient la description d'une classe d'objets.

La différence entre API et ALI ressort immédiatement de leur nom. L'API offre une interface de programmation alors que l'ALI gère les objets. L'API et l'ALI sont tous les deux situés entre le processus d'application et la couche 7, mais leur tâche est tout à fait différente. Et cependant, parfois dans la littérature, l'API est appelé ALI (réf. /10/).

En conclusion, on peut dire que le Profibus est mieux adapté pour les réseaux de terrain que Mini-MAP et que MAP.

Ceci est dû au fait que MAP est trop lent et trop cher (en infrastructure, connexion et en temps) et que Mini-MAP est fortement controversé.

L'argument que MAP est trop cher est un point fondamental contre l'utilisation de MAP pour les réseaux de terrain. Aux niveaux 1 et 2, l'intérêt majeur est d'automatiser un processus par des automates programmables, par exemple, et non de communiquer. La communication ne vient qu'en second lieu.

4. L'architecture utilisée par la SA SIEMENS[28]

Les chapitres 2 et 3 présentent surtout des standards et des normes existants en matière de réseaux locaux industriels. Nous avons notamment parlé de MAP/MMS, de Mini-MAP et des stations EPA (au chapitre 2). Au chapitre 3, nous nous sommes efforcés de donner un aperçu du réseau de terrain PROFIBUS, développé en Allemagne.

A côté de ces normes nationales et internationales, différentes normes propres à une firme existent dans le monde entier. Ces normes, dites normes privées, subsistent encore, malgré l'existence de produits opérables dans les normes nationales et internationales.

Essayons donc, dans ce chapitre, de présenter une norme privée. Pour ce faire, nous avons choisi de présenter les architectures de réseau industriel développées par la SA Siemens : les architectures SINEC (*Siemens Network Architecture*). Nous terminerons ce chapitre par une comparaison entre SINEC et MAP/MMS, Mini-MAP et PROFIBUS.

4.1. Présentation des architectures SINEC

SINEC a été développé comme un protocole ouvert adoptant, là où c'était possible, des normes existantes telles que MMS, sauf dans les cas où les normes n'étaient pas encore prêtes lors de la spécification. Dans ces cas, des solutions particulières ont été choisies. Mais dès que possible, la mutation vers des systèmes MAP/MMS, .. est prévue.

Nous commencerons donc par une présentation des architectures SINEC. Ensuite, nous parlerons des services offerts par la couche application et nous terminerons cette partie du quatrième chapitre par un essai de justification d'une telle multitude d'architectures, qui ne sont pas nécessairement compatibles avec les normes nationales et internationales.

Les architectures SINEC

La SA Siemens propose à sa clientèle un ensemble d'architectures différentes. Chaque architecture possède ses spécificités, s'adresse à une clientèle spécifique et se situe à un niveau précis dans la hiérarchie des réseaux industriels locaux.

28 Nous nous sommes basés principalement sur les références /16/, /17/, /18/ et /20/.

Après la simple présentation des différentes architectures, nous allons expliquer brièvement les couches et les principes de SINEC qui offre un protocole ouvert pour la communication industrielle.

Nous aborderons dans l'ordre SINEC H1, SINEC H2, SINEC L2, SINEC L1 et SINEC H3.

SINEC H1

La première architecture proposée est la SINEC H1 (*fig. 4.1. L'architecture de Sinec H1*). Il s'agit d'un réseau local industriel basé sur le CSMA/CD (IEEE 802.3) en bande de base permettant une vitesse de transmission maximale de 10 Mbits/s.

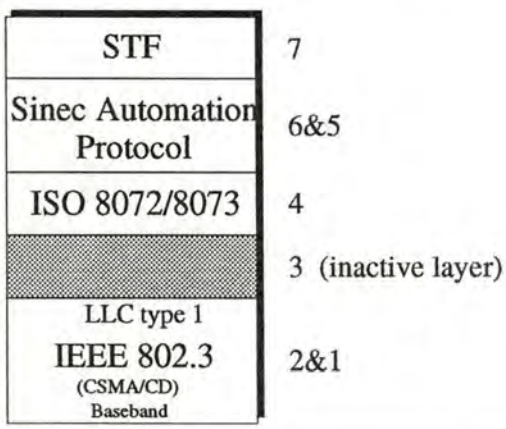


fig. 4.1. L'architecture de Sinec H1

La sous-couche LLC est de type 1 (IEEE 802.2). La couche 3 est un *inactive subset* et le protocole de transport ISO (ISO 8072/8073) se situe à la couche 4.

SINEC utilise des protocoles spécifiques aux couches 5, 6 et 7. Nous reviendrons plus en détail sur ces couches dans la suite. Puisque les couches 3 à 7 sont à peu près identiques pour les différentes architectures SINEC, on ne les mentionnera pas chaque fois.

SINEC H2

Le réseau H2 existe en plusieurs variantes. On trouve, d'une part, le H2b, un réseau *broadband* à 10 Mbits/s [et donc comparable à MAP] et, d'autre part, le H2c, réseau *carrierband* à 5 Mbits/s [et donc comparable à Mini-MAP] (*fig. 4.2. L'architecture SINEC H2b et H2c*).

Les réseaux H2 utilisent la méthode d'accès déterministe par token pour accéder au bus. Le réseau H2 est donc réalisé sur un token bus (IEEE 802.4).

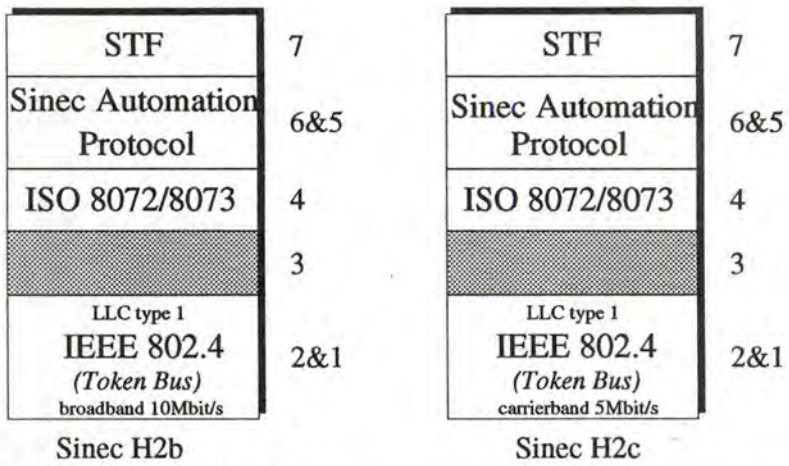


fig. 4.2. L'architecture SINEC H2b et H2c

SINEC L2

Sinec L2 est le surnom d'un ensemble de produits du *low-cost*, produits qui ont le même but mais qui sont destinés à des applications différentes. On quitte donc les niveaux supérieurs de la hiérarchie des réseaux locaux industriels pour arriver, avec le L2, aux réseaux de terrain. Il existe des variations L2 pour MMS, FMS, STF et pour des services spécifiques.

A ce niveau, Siemens a opté pour une architecture qui utilise la norme PROFIBUS partie 1 (DIN 19245/1) pour les couches 1 et 2 alors que pour la couche 4, Siemens élabore une couche transport nouvelle appelée L2-Transport.

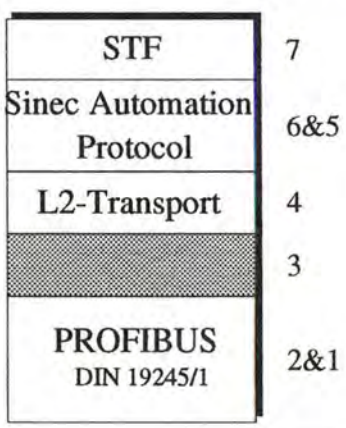


fig 4.3. L'architecture SINEC L2

Aux couche 5 à 7, Siemens offre plusieurs possibilités telles que STF, MMS, FMS et autres (*fig 4.3. L'architecture SINEC L2*).

Expliquons à présent plus en détail les différentes couches. Nous allons nous concentrer ici sur les couches 5 à 7, couches qui diffèrent des normes nationales et internationales.

Les couches 5 et 6 sont reprises par Sinec AP (*Automation Protocol*) qui utilise directement les services de la couche transport. Des règles strictes de codage et de décodage ont été fixées pour l'efficacité du transfert de messages. Le déroulement du protocole est contrôlé par le AP-Monitor.

A la couche application se situe l'élément de service STF (*Sinec Technologische Funktionen*). Il s'agit d'un *subset* (sous-ensemble) des services offerts par MMS (IS 9506/1). STF contient l'ensemble des services utiles pour l'automatisation. Les services les plus importants sont la gestion des variables, des domaines et des instances de programmes ainsi que quelques services généraux du genre *Get_Status*.

Au moment du développement de SINEC, MMS, qui offre des services équitables, n'était pas encore assez stable et Siemens n'espérait pas que MMS allait s'imposer rapidement. La solution consistait alors de développer STF. Lors de l'apparition de la version définitive de MMS, on a revu et adapté STF en fonction de MMS.

La même remarque peut être faite pour l'élément de service FTS (*File Transfer Service*). FTS correspond de par son concept à FTAM avec son *virtual filestore*. Mais FTS offre un codage/décodage plus efficace. Pour ce faire, FTS utilise, aux couches 5 et 6, deux normes ouvertes de Siemens.

Ici nous n'aborderons pas les principes généraux de STF puisqu'ils correspondent presqu'entièrement à ceux de MMS. Pour une approche plus approfondie, nous recommandons de lire les références /16/, /17/ et /18/.

A côté des éléments de services d'application déjà mentionnés, l'architecture SINEC (comme d'ailleurs toutes les autres architectures déjà expliquées) offre une gestion de réseau sur laquelle nous ne nous étendrons pas ici.

En plus des protocoles SINEC, les réseaux SINEC permettent à l'utilisateur d'employer d'autres protocoles (tels que TCP/IP, Novell, NFS, ...) sur un même noeud de connexion à côté des services Sinec.

La figure 4.4. donne une synthèse des architectures SINEC.

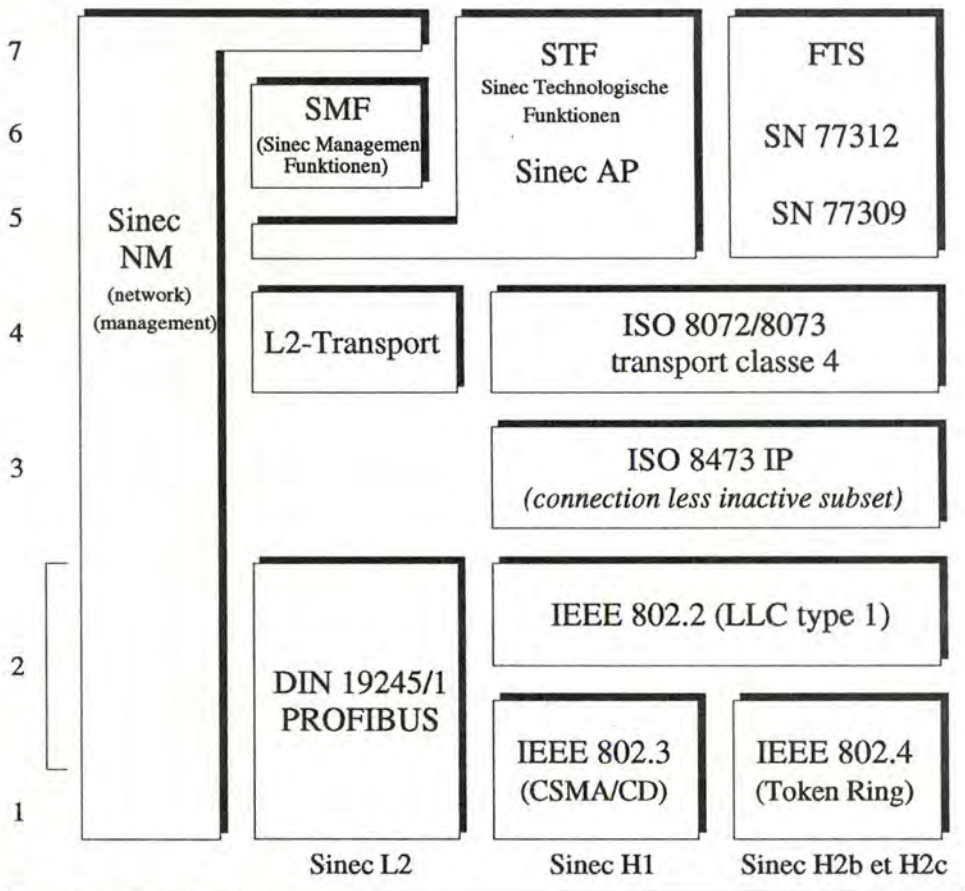


fig. 4.4. Les architectures Sinec

Les services offerts

Les services qui nous intéressent ici sont les services offerts par la STF à l'utilisateur de la STF. Puisque STF est un subset de MMS, on retrouve évidemment tout un ensemble de services de MMS.

Ce qui distingue les services offerts par STF des services offerts par MMS c'est que STF ne contient que les ensembles fonctionnels et les services désirés ou utilisés en ce moment.

En plus des services comparables à MMS, STF offre aussi un ensemble de services nécessaires pour pouvoir communiquer avec des stations et des réseaux anciens, présents avant la définition de STF.

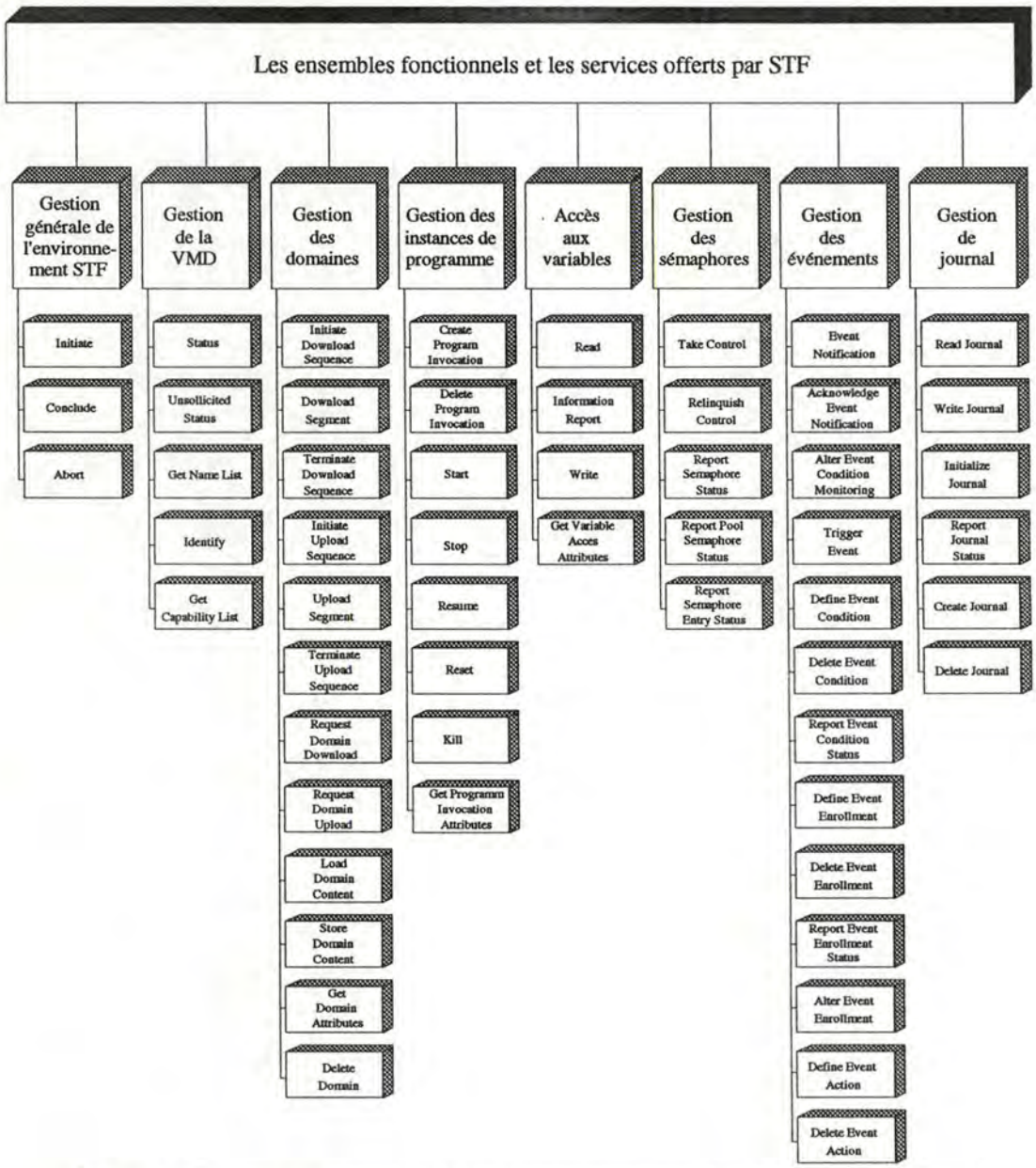


fig. 4.5. Les ensembles fonctionnels et les services offerts par STF

Les ensembles fonctionnels ainsi que les services offerts par STF sont repris à la figure 4.5.. Pour faciliter la comparaison avec les services de MMS, les noms sont mis en anglais. La liste des noms en allemand se trouve à l'annexe B. C'est aussi à l'annexe B que se trouvent les services non ouverts, non repris à la figure 4.5..

Les raisons d'une telle architecture

Après avoir vu les différentes architectures proposées par Siemens, nous pouvons nous demander pourquoi Siemens a développé et développe encore des standards qui lui sont propres (mais quand même ouverts) alors que des standards nationaux et internationaux existent. Une partie des raisons a déjà été mentionnée aux points 1 et 2 du paragraphe 4.1..

- Il est nécessaire et même obligatoire de pouvoir communiquer avec des stations anciennes qui ne correspondent à aucun standard ouvert.
- Toutes les stations développées par Siemens doivent être facilement interconnectables même à travers des réseaux différents. Il est donc essentiel à partir d'un certain niveau d'utiliser des architectures identiques .
- Les standards sont bons et nécessaires, mais dans des cas précis, on a besoin d'une spécialisation et la connexion est alors réalisée par *gateway*.
- Un standard accepté par tout le monde doit offrir (presque) toutes les possibilités imaginables. Mais souvent les clients ne veulent pas acheter ce dont ils n'ont pas besoin. Ainsi Siemens n'offre que ce qui est désiré par la clientèle.
- La multiplicité de normes SINEC se justifie par le fait que chaque norme est prévue pour un niveau différent ou pour une utilisation différente.
- Une dernière raison qu'on évoquera ici est que SINEC a été développé parce qu'on ne s'attendait pas chez Siemens à ce que MMS arrive dans des délais respectables. Mais dès l'apparition de MMS, Siemens a revu sa structure au niveau 7 et l'a adaptée en fonction de MMS afin d'offrir un subset compatible.

4.2. Comparaison avec MAP/MMS et PROFIBUS

Pour terminer ce chapitre, nous allons essayer de faire une comparaison entre les architectures fournies par Siemens et les normes nationales/internationales. Une analyse approfondie et en détail des différents standards demanderait un temps trop long et n'est pas l'objet de cette partie.

Nous allons essayer de comparer ce qui est comparable. Ce qui signifie que la comparaison s'effectue entre les réseaux d'un même niveau dans la hiérarchie des réseaux vue au chapitre 1. Nous opposerons donc le Sinec H1 et les Sinec H2 à MAP/MMS et le Sinec L2 à Profibus. Nous laisserons de côté le Sinec L1 qui n'a

aucun correspondant national ou international et le Sinec H3 qui n'est utilisé que comme *backbone* (pour l'interconnexion de réseaux) auquel on ne peut pas directement connecter des stations.

Sinec H1 et H2 comparé à MAP/MMS

La comparaison entre les réseaux H1 et H2 et MAP/MMS peut se faire d'une part en se basant sur les architectures et d'autre part, en se basant sur l'élément de service d'application majeur respectivement STF et MMS.

La distinction entre les deux architectures est directement visible. Toutes deux comportent une division en plusieurs couches, mais alors que MAP utilise des standards agréés par l'ISO, Siemens utilise ses propres normes aux couches 5,6 et 7. La couche transport est la même dans les deux cas (ISO 8072/8073 classe 4).

Pour MAP, les 7 couches sont toutes fonctionnelles. Mais dans l'architecture Sinec, la couche 3 n'est pas utilisée actuellement. Elle est bel et bien présente, mais uniquement comme *connection less inactive subset.* A l'avenir, il sera donc possible d'ajouter, sans grandes difficultés, une couche réseau fonctionnelle dans les architectures Sinec.

Aux couches 1 et 2 Sinec et MAP sont identiques. Le Sinec H2 ne figurera plus dans la gamme des réseaux, entre autre pour cette raison. Le Sinec H2 est remplacé par MAP.

Le H1, par contre, qui est un réseau Ethernet (CSMA/CD) reste toujours disponible. Son avantage est que beaucoup d'entreprises ont déjà un réseau comparable.

Consacrons maintenant un peu de place à la comparaison entre les deux éléments de services d'application STF et MMS.

STF est comme nous l'avons déjà dit un subset de MMS et ne reprend de ce fait que quelques services de MMS. Par ailleurs, STF offre des services spécifiques pour la communication avec des anciennes stations.

MMS offre au programmeur la possibilité d'utiliser une interface de haut niveau appelée API. Le correspondant pour SINEC s'appelle SCI (*Sinec Communication Interface*).

STF et MMS ont les mêmes principes de base tels que les objets, la VMD, les variables, les domaines, les instances de programme, ... Il n'y a que des différences minimes entre la description des objets. Il sont en gros les mêmes. Dans le cas où

on trouve une différence, celle-ci est surtout due au fait que STF n'est qu'un subset de MMS.

Mais outre des attributs identiques à ceux dans MMS, chaque description d'objet comporte un attribut local *Lage* qui permet de donner la localisation précise du contenu de la mémoire en fonction de la méthode de mémorisation.

Les différences au niveau du protocole ne sont pas traitées dans ce travail.

Sinec L2 comparé à Profibus

Une comparaison entre les architectures du L2 et du Profibus s'avère facile. Les deux architectures disposent des couches 1,2 et 7, mais celle du L2 offre en plus une couche 4, absente dans le Profibus. Les couches 1 et 2 des deux architectures sont définies dans DIN 19245/1 Profibus. De même, on retrouve à la couche 7 dans les deux cas un subset de MMS qui n'est pourtant pas le même dans les deux cas.

La couche 4 du L2 se distingue de la couche 4 ISO et est adaptée à l'environnement industriel et aux réseaux de terrain. Nous étudierons plus en détail une telle couche transport au chapitre 5.

Lorsqu'on compare STF et FMS, on s'aperçoit d'abord, que les services offerts constituent un sous-ensemble des services offerts par MMS. Mais chacun des deux offre en plus un ensemble fonctionnel qui lui est propre. STF offre ainsi des services qui permettent de communiquer avec des stations anciennes et FMS offre des services de gestion du répertoire des objets.

STF et FMS ont à peu près les mêmes objets, mais les choix sont beaucoup plus limités pour FMS dans les options des attributs.

De plus, le Profibus possède l'ALI qui joue le rôle d'interface entre l'application et la couche application.

Ainsi se termine la présentation des architectures Sinec. La couche application, identique pour les architectures, permet une interconnexion facile des différents équipements et réseaux.

De plus, les architectures offerts par Sinec permettent d'utiliser des types de réseaux différents sans pour autant devoir changer fondamentalement les programmes. Ce qui est naturellement intéressant pour les utilisateurs et les programmeurs.

La suppression de Sinec H2 marque le début d'une nouvelle ère qui se poursuivra dans le futur. La mutation des produits Sinec vers des standards nationaux et internationaux va encore s'accélérer. Déjà maintenant des *gateway* vers les standards sont disponibles, mais ces solutions ne sont que très peu (pour ne pas dire pas du tout) demandées par la clientèle.

5. Le projet SPS90-Transport

Dans les chapitres précédents, nous nous sommes principalement occupés des standards de réseaux locaux industriels tant au niveau international (MAP) qu'au niveau national (PROFIBUS) et privé (SINEC). Nous avons présenté et comparé les différents standards existants.

Ce cinquième chapitre aborde un problème plus pratique, à savoir l'élaboration d'une couche transport pour un équipement d'automation, à savoir pour un automate programmable. Ceci était aussi mon occupation majeure durant mon stage chez Siemens Amberg. Je me suis d'abord introduit dans l'environnement de travail : Profibus, Sinec L2, le cahier de charges existant. Par après, j'ai consulté d'autres travaux et surtout la référence /14/. Des réflexions sur la gestion interne ont suivi cette première phase. Dans la suite, j'ai participé au développement des diagrammes séquence-temps, du diagramme états-transitions et des diagrammes SDL. La durée du stage était trop courte pour pouvoir participer à l'implémentation et aux tests.

L'idée de la SA Siemens est de mettre sur le marché une ligne de produits d'automates programmables appelés SPS90. Cet automate est prévu pour le réseau Sinec L2 et son architecture de communication reprend donc, comme nous l'avons vu au chapitre précédent, les couches 1, 2, 4 et 7 du modèle OSI. La SPS90 est une simplification de l'architecture dans le cadre du développement pour permettre l'inclusion dans la communication d'appareils de bas de gamme .

La SPS90 doit être conçue de manière à répondre aux objectifs fixés, objectifs parmi lesquels nous trouvons la simplicité, la rapidité et le coût. De ce fait, les couches 4 et 7 sont réduites au strict minimum ce qui nous donne l'architecture montrée à la figure 5.1..

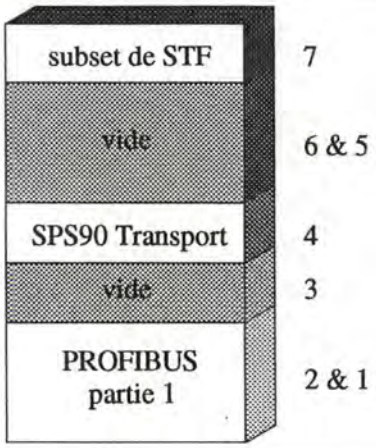


fig. 5.1. L'architecture de la SPS90

Avec la SPS90, Siemens introduit une nouveauté dans les équipements d'automatisation, l'interface multipoint.

Autrefois, lorsqu'on voulait entrer en communication avec un automate programmable, le seul moyen d'y arriver était une ligne de communication directe (1 à 1) entre l'appareil de programmation (*Programmiergerät* - PG) et cet automate programmable (*fig. 5.2. Communication 1 à 1*)

fig. 5.2. Communication 1 à 1

La SPS90 introduit maintenant ce qu'on appelle une interface entre PG et l'automate programmable. Par cette interface, plusieurs PG's peuvent communiquer avec un même automate programmable. De plus, un PG a la possibilité d'entrer en communication avec plusieurs automates. La figure 5.3. visualise cette idée d'interface multipoint.

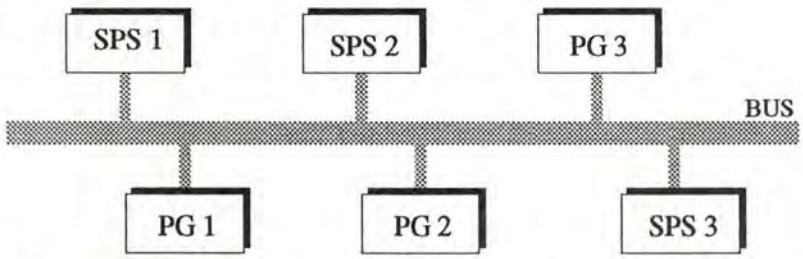


fig. 5.3. Schéma de l'interface multipoint

Après avoir introduit ainsi rapidement l'idée de cet automate programmable, nous pouvons nous concentrer sur la couche transport de la SPS90, l'objet du présent chapitre.

Pour fixer les idées, nous allons faire un rappel de la couche transport ISO (ISO 8072/8073) avant de donner les raisons de la conception d'une nouvelle couche transport. Nous terminerons ce chapitre en étudiant plus en détail la couche transport pour la SPS90.

5.1. Rappel des fonctions et services de la couche transport standard ISO 8072/8073[29]

Avant d'entrer dans une démarche de conception d'une nouvelle couche transport, il nous paraît fort utile de nous rappeler la structure et l'utilité ainsi que les fonctions d'une couche transport. Pour ce faire, nous prenons comme exemple la couche transport définie par l'ISO.

La couche transport ISO concernée est définie dans les normes ISO par deux papiers différents :

IS 8072 qui définit les services et

IS 8073 qui parle plus du protocole.

Cette couche correspond à la couche 4 du modèle OSI (*fig. 5.4. La couche 4 dans le modèle OSI*).

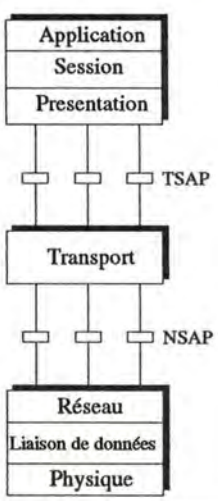


fig. 5.4. La couche 4 dans le modèle OSI

29 Nous nous sommes basés principalement sur les références /7/ et /8/

C'est la couche transport qui est la première des sept couches du modèle à permettre une communication *end-to-end* (de bout en bout).

Différentes classes de protocoles sont définies par la norme :

classe 0 : *simple class* : la classe 0 est la classe la plus simple et elle est tout à fait compatible avec la recommandation T.70 du CCITT pour les terminaux télétex. La classe 0 est prévue pour l'utilisation avec des réseaux de type A[30].

classe 1 : *basic error recovery class* : la classe 1 a un *overhead* minimal et elle permet de récupérer des erreurs signalés provenant du réseau. La classe 1 est prévue pour l'utilisation avec des réseaux de type B.

classe 2 : *multiplexing class* : la classe 2 permet le multiplexage de plusieurs connexions transport sur une seule connexion réseau. Elle est prévue pour une utilisation avec un réseau de type A. Elle offre, en plus, le choix entre un contrôle de flux explicite ou pas de contrôle de flux.

classe 3 : *error recovery and multiplexing class* : la classe 3 reprend les caractéristiques des classes 1 et 2. Elle est prévue pour l'utilisation sur un réseau de type B.

classe 4 : *error detection and recovery class* : la classe 4 offre les mêmes caractéristiques que la classe 3 mais possède en plus des possibilités de détection et de récupération d'erreurs résultant d'une couche réseau moins sûre, y compris les réseaux CLNS*. De plus, elle est plus performante dans la récupération des erreurs provenant du réseau lui-même. La classe 4 est prévue pour l'utilisation avec des réseaux de type C.

Le choix de la classe dépend donc en grande partie de la qualité du réseau utilisé.

La norme définit aussi deux types de couche transport. Une première qui fournit un service *connection oriented* (COTS*) et une seconde qui fournit un service *connection less* (CLTS*). De plus, la couche 4 permet de réaliser le passage d'une

30 On distingue différents types de réseaux :
type A : réseaux avec taux d'erreurs résiduels acceptables et avec taux d'erreurs signalés acceptables.
type B : réseaux avec taux d'erreurs résiduels acceptables mais avec taux d'erreurs signalés inacceptables.
type C : réseaux avec taux d'erreurs résiduels inacceptables.

couche réseau *connection less* à une couche transport *connection oriented.* Pour ce faire, il est nécessaire d'utiliser un protocole de transport de classe 4 afin de pouvoir fournir la sécurité minimale souhaitée.

Comme toutes les autres couches du modèle OSI, la couche 4 offre aussi des services à son utilisateur (la couche 5 en l'occurrence). Ces services sont repris au tableau 5.1.. Il va de soi que les services de connexion et de déconnexion ne sont disponibles que pour la version *connection oriented* du protocole.

T_Connect.	request indication response confirmation
T_Data.	request indication
T_Expedited_Data.	request indication
T_Disconnect.	request indication

tableau 5.1. Les services offerts par la couche transport

La présence d'une couche transport est justifiée par le fait qu'elle ajoute quelque chose au système de communication. Comme nous l'avons déjà mentionné, elle permet de faire le passage entre un protocole *connection less* à la couche 3 et un protocole *connection oriented* à la couche 4[31]. Outre cette fonction très intéressante, la couche transport offre en plus diverses fonctionnalités :

- **Des fonctionnalités offertes à tout moment** : multiplexage et démultiplexage, détection d'erreurs, reprise d'erreurs (*error recovery*)
- **Des fonctionnalités de connexion et de déconnexion.** Ces fonctionnalités ne sont présentes que pour la version *connection oriented* du protocole.
- **Des fonctionnalités de transfert de données** : l'envoi de TPDU, concaténation et séparation, segmentation et réassemblage, *splitting* et *recombining*, contrôle de flux par le mécanisme de la fenêtre glissante, *expedited data.*

Ces fonctionnalités diffèrent d'une classe à l'autre. Seule la classe 4 les intègre toutes et a par conséquent le plus grand *overhead.* C'est aussi cette classe qui est

31 L'inverse est aussi pensable, mais n'a pas de sens en pratique.

encore généralement utilisée pour les réseaux locaux puisque les réseaux sont encore souvent de type C, c'est-à-dire avec taux d'erreurs résiduels inadmissibles.

D'autres fonctionnalités non reprises dans la première édition (du 15-07-86) de la norme sont par exemple, : l'encryptage, la facturation, l'échange et le contrôle du status de QOS (*Quality Of Service*), le blockage, le relâchement temporaire d'une connexion réseau, etc..

Ainsi s'achève ce bref rappel de la fonctionnalité de la couche transport ISO. Nous n'avons guère parlé du protocole, mais ce n'était pas non plus notre objectif.

Dans la suite de ce chapitre, nous allons nous préoccuper plus en détail d'une nouvelle couche transport en élaboration chez Siemens.

5.2. Motivation pour une nouvelle couche transport

Lorsqu'on décide de développer une nouvelle couche transport, il faut avoir des raisons fondées pour le faire car le développement d'une couche 4 demande du temps et est assez compliqué. De plus, des couches transport sont disponibles sur le marché et l'achat d'une telle couche est moins onéreux que le développement d'une nouvelle couche.

Une telle couche est, par exemple, l'ISO 8072/8073 que nous venons de présenter. Mais comme il s'agit d'une couche universelle, elle est très complexe et coûteuse en temps et en espace mémoire.

Donc, elle ne convient pas pour la SPS90 pour laquelle l'espace mémoire prévu pour la couche 4 n'est que de 5 KB (Kilo byte). En outre, la couche transport utilisée doit convenir aux exigences des réseaux de terrain. Aux réseaux de terrain, on exige souvent des temps de réaction minimaux.

On peut reprocher à l'ISO 8072/8073 d'être trop volumineuse, trop lente lors de l'établissement de la connexion puisqu'il faut toujours négocier et enfin trop lente en exécution. L'ISO 8072/8073 est trop complexe et offre des fonctionnalités dont on n'a pas besoin.

On n'a pas pu prendre la couche 4 du PROFIBUS puisque celui-ci n'en possède pas et que Sinec se base sur une couche 4. On a de plus voulu être compatible avec

les autres architectures Sinec aux couches 5 à 7 de manière à permettre une communication aisée et rapide entre les différents systèmes.

On peut naturellement se demander pourquoi on ne prend pas carrément le Profibus sans la couche 4. C'est surtout le remplacement incomplet des couches 3 à 6 qui rend une telle solution insuffisante. Le manque de fonctionnalités performantes pour le travail *end-to-end* et pour le contrôle de flux sont le plus grand reproche qu'on puisse faire à cet égard.

5.3. Démarche de conception d'une couche transport

Ce chapitre a pour but, comme nous l'avons déjà dit, de décrire une démarche de conception d'une couche transport. A cet égard, il est très utile de présenter d'abord une fois l'environnement dans lequel l'échange de messages se fait. Ensuite, nous parlerons des deux étapes de conception qui sont la spécification et l'implémentation.

5.3.1. L'environnement de communication

Le but du jeu est de concevoir une couche transport pour l'automate programmable nommé SPS90. Il s'agit d'un équipement d'un réseau de terrain. Nous ne devons donc surtout pas oublier qu'à un niveau si bas, on sait toujours qui va communiquer avec qui. Ce qui est, par exemple, tout à fait différent pour des WAN (*Wide Area Network*) où l'on peut communiquer avec à peu près tout le monde.

De plus, la tâche principale de cet automate programmable est, par exemple, le contrôle d'une partie d'un processus de fabrication. La communication ne vient qu'en second lieu.

Pour réaliser les tâches de contrôle et de communication, la SPS90 dispose entre autres d'un système d'exploitation multitâches appelé AMOS (*Advanced Multitasking Operating System*). Tous les programmes lancés sur cette SPS90 tournent au-dessus d'AMOS. La communication est composée de trois tâches formées respectivement par les couches 1/2, 4 et 7. Expliquons maintenant brièvement le concept d'AMOS.

AMOS est, comme nous venons de le dire, un OS multitâches. Donc, plusieurs tâches (pour la SPS90 leur nombre est limité à 15) peuvent s'exécuter quasi simultanément. Le temps CPU est divisé entre les différentes tâches. Chaque tâche est à chaque moment dans un des états désignés à la figure 5.5.. Cette figure montre aussi les passages possibles d'un état vers un autre. Les flèches pointillées montrent des changements d'états possibles, mais peu probables.

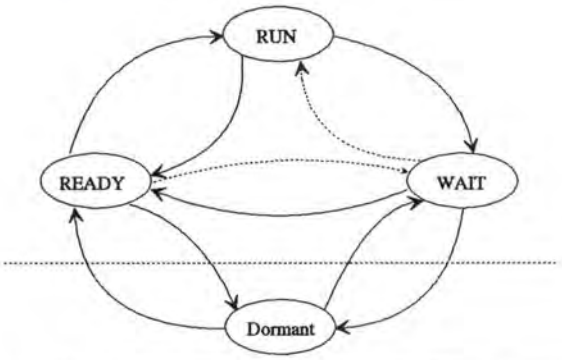


fig. 5.5. Les états possibles d'une tâche au dessus d'AMOS

La tâche dans l'état *RUN* est celle qui est exécutée actuellement. Il n'y a qu'une tâche à la fois dans l'état *RUN* . Les tâches en état *READY* sont prêtes à être exécutées. En état *WAIT*, la tâche attend l'arrivée d'un message ou d'un événement. Elle n'est pas prête à être exécutée. L'état *DORMANT* représente une désactivation de la tâche. La tâche reste en mémoire, mais elle n'est plus connue par AMOS et on ne lui attribue plus aucun temps CPU.

Pour pouvoir communiquer, les tâches s'échangent des messages stockés dans une *mailbox*. La *mailbox* est formée par une liste circulaire traitée par le principe FIFO. Dans le cas, où toutes les tâches sont dans l'état *WAIT* ou dans l'état *DORMANT*, un tâche *idle*, dont la fonction est de ne rien faire, est exécutée jusqu'à ce qu'une autre tâche passe à l'état *READY* (*fig. 5.6. L'échange de messages entre les différentes tâches*).

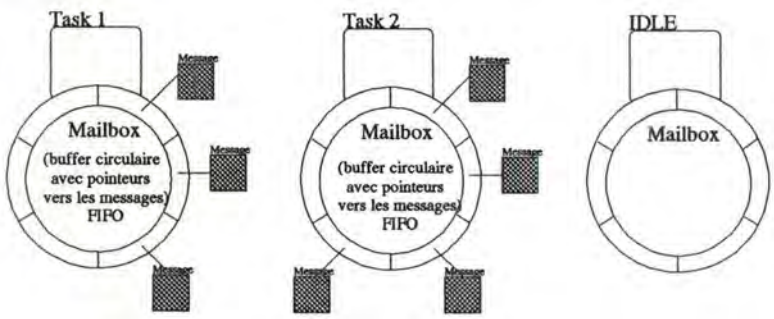


fig. 5.6. L'échange de messages entre les différentes tâches.

Une tâche reste à l'état *RUN* jusqu'au moment où, soit un compteur passe à 0 et la tâche passe à l'état *ready*, soit où la tâche doit attendre qu'un événement survienne et elle passe en conséquence à l'état *WAIT*. Dans les deux cas, on lance un *scheduler* et, en dépendance des priorités et de l'état des tâches, on en sélectionne une qui passe à l'état *RUN*.

Chaque tâche a une certaine priorité. En ce qui concerne la communication, c'est la couche 1/2 qui représente la plus grande priorité suivie de la couche 4 qui, elle, possède une priorité plus grande que la couche 7.

La couche 4 reste dans l'état *RUN* jusqu'à ce qu'il y ait

- interruption de la part du système d'exploitation
- envoi d'un message à une tâche plus prioritaire
- passage à 0 d'un compteur.

Nous pouvons à présent donner l'architecture de la SPS90 avec plus de détails sur les couches (*fig. 5.7. L'architecture d'une SPS90*).

La couche 1/2, appelée AMPRO2, est compatible PROFIBUS partie 1 (DIN 19245/1). La couche 4, AMPRO4, est celle que nous décrirons et la couche 7 est formée par AMPRO7 , une simplification de STF adaptée aux besoins spécifiques de la SPS90. C'est surtout l'*overhead* qui a été réduit. La communication avec d'autres stations Sinec n'est par conséquent possible que par un *gateway* qui est pourtant très simple.

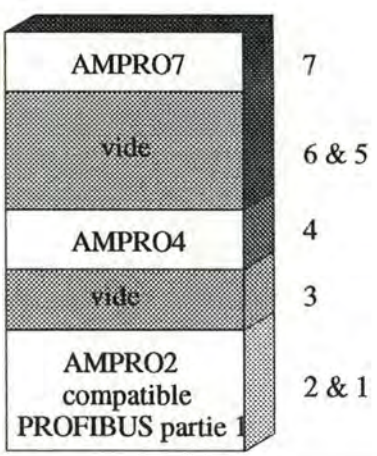


fig. 5.7. L'architecture d'une SPS90

5.3.2. La spécification

Comme pour tout développement informatique, la spécification est la partie principale d'une conception d'une couche transport. Il s'agit de définir en détail les interfaces ainsi que le comportement externe et interne de la couche.

Pour cette raison, on peut diviser cette partie du cinquième chapitre en deux. D'une part, on a le comportement externe de la couche caractérisé par les services et le protocole. D'autre part, on a son comportement interne, surtout formé par la gestion interne de la couche 4.

Dans la spécification, nous utiliserons différents moyens de présentation (des diagrammes séquence-temps, des diagrammes états-transitions et des diagrammes SDL, par exemple). Ces présentations ont pour seul but de mieux faire comprendre et de visualiser le problème et les solutions choisies. Il n'y a en aucun cas un automatisme qui permet de passer d'une présentation à une autre.

Toute la spécification que nous développerons ici, doit être considérée comme la présentation d'une démarche. En raison du cadre limité, on n'a pas pu reprendre dans un mémoire l'entièreté d'une spécification d'une couche transport. Il est donc également clair que la démarche que nous présentons ne peut qu'aborder des problèmes rencontrés en réalité. De plus, le choix de simplifier certaines parties nous permet de présenter des étapes moins lourdes et moins complexes. Une telle simplification sera surtout utilisée lors de la réalisation des diagrammes SDL, diagrammes dans lesquels une présentation de tous les cas d'erreurs, par exemple, demanderait autant de place qu'un travail complet.

La démarche que nous présentons ne prétend en aucune manière être exhaustive (complète) mais elle sert à illustrer le développement d'une couche transport dans un environnement précis et différent de la couche 4 ISO. Les différentes phases du développement sont :

- l'établissement du cahier des charges,
- la consultation d'autre travaux,
- le développement de la gestion interne,
- l'établissement des diagrammes séquence-temps,
- l'établissement des diagrammes états-transitions et
- l'établissement des diagrammes SDL

LE CAHIER DES CHARGES

Le cahier des charges se trouve au début de tout développement informatique. Ce cahier est important et nécessaire puisqu'il fixe les limites de travail et permet ainsi de partager le travail entre les différents membres d'un groupe.

Comme nous n'avons pas la possibilité de reprendre tous les détails du cahier des charges - ce qui ne serait d'ailleurs ni utile et ni intéressant - nous parlerons surtout des services et un peu du protocole.

Rappelons rapidement les idées de base du développement de la AMPRO4. Elle doit être petite, rapide et simple. Ces idées ont leur répercussion dans tout ce qui va suivre. La couche 4 développée est donc une couche transport rudimentaire qui ne sait pas faire trop de choses, mais ce qu'elle fait, elle le fait vite.

Dans une première phase de développement, la AMPRO4 n'est prévue que pour des stations actives. De plus, une station, qui a pour couche 4 la AMPRO4, ne peut jamais être à l'origine d'une connexion. Il n'y a donc pas de communication directe possible entre deux stations SPS90 dans une première phase. Il est encore important de remarquer que les différentes relations de communication sont préprojetées. Chaque station sait donc avec qui et comment elle peut communiquer.

Pour fixer les idées, citons différentes contraintes spécifiques à la AMPRO4:

- La AMPRO4 n'offre pas la possibilité de segmenter et de réassembler des données. Si une segmentation et un réassemblage s'avèrent indispensables, ceux-ci doivent être réalisés à un niveau supérieur. Les messages envoyés à la station au niveau 4 sont des messages clôturés en eux-mêmes (*Send_EOM*) et envoyés à la station paire.

- La AMPRO4 travaille *connection oriented.* Il faut en conséquence toujours ouvrir une connexion (*Connect*) avant de pouvoir envoyer des données (*Send_EOM*). Ensuite, il faudra refermer la connexion (*Disconnect*).

- La AMPRO4 offre un contrôle d'erreurs. Pour quelques erreurs, la AMPRO4 demande la retransmission du message (et cela un nombre limité de fois). Pour d'autres erreurs, notamment des erreurs de protocole, la

AMPRO4 va initier une déconnexion. La réalisation du contrôle d'erreurs est la chose la plus délicate puisqu'il faut prévoir toutes les erreurs possibles et ceci avec le moins de codes possibles.

- La AMPRO4 doit aussi pouvoir communiquer avec d'autres stations et notamment avec des stations plus anciennes et des stations plus puissantes. Ainsi, elle doit, par exemple, reconnaître des messages tels que IR (*Immediate Response*), *IDLE*, *XOFF*,[32] ... bien qu'elle ne les envoie jamais.

- Dès que la connexion est ouverte, chacune des deux stations peut envoyer des données à la station paire. Chacune des deux stations peut également initier la déconnexion.

- Le contrôle de flux (*flow control*) offert par la AMPRO4 est un contrôle rudimentaire. Il n'existe pas de mécanisme de fenêtre glissante (*sliding window*).
  Au départ, chacune des deux stations en communication met à disposition un *buffer* qui peut contenir un message (*credit* = 1) et le dit à la station paire. Le *buffer* est utilisé pour stocker un message envoyé par l'autre station. Lors de l'acquittement de la réception du message, on envoie à nouveau un XON (*credit* = 1) avec la ACK-TPDU pour indiquer qu'il y a à nouveau un *buffer* qui est prêt. La ACK-TPDU n'est envoyée que lorsqu'il y a vraiment un nouveau *buffer* qui est mis à disposition.
  Ceci ne retient naturellement pas le AMPRO4-*user* de demander plusieurs services *T_Send_EOM* l'un après l'autre sans attendre l'acquittement du premier. C'est à la couche 4 de gérer ces *T_Send_EOM* services. (Le nombre de *T_Send_EOM* est cependant limité, soit par la spécification, soit par la place mémoire disponible.)

Le cahier des charges définit aussi la structure exacte des PDU et les valeurs possibles des différents champs de la PDU. Nous n'entrerons pas plus dans les détails du protocole.

Ce qui nous préoccupe un peu plus est l'interface avec l'utilisateur de la couche 4 (*layer 4 user*) qui est la AMPRO7 dans notre cas précis. Le tableau 5.2. donne la

32 IDLE est une PDU qui indique qu'il n'y a actuellemnt pas de message mais qu'on ne coupe pas la connexion. XOFF correspond à une remise à 0 explicite du crédit.

liste des services offerts, une brève explication et un indicateur pour donner la station qui l'utilise (I = Initiateur et R = Répondeur).

T_Open.req/.con	Préparation à l'ouverture et attribution d'une CN_ID si possible.	I/R
T_Close.req/.con	Fermeture définitive de la connexion avec libération de la CN_ID.	I/R
T_Connect.req/.con	Demande d'ouverture d'une connexion. (Pas encore possible pour la AMPRO4.)	I
T_Await_Connect.req/.con	Attente de la demande d'ouverture de la station paire.	R
T_Accept.req/.con	Acceptation de la demande d'ouverture de la station paire.	R
T_Reject.req/.con	Rejet de la demande d'ouverture de la station paire.	R
T_Receive.req/.con	Préparation à la réception d'un bloc de données.	I/R
T_Send_EOM.req/.con	Demande d'envoi de messages.	I/R
T_Disconnect.req/.con	Demande de déconnexion.	I
T_Await_Disconnect.req/.con	Attente d'une demande de déconnexion de la part de la station paire.	R

tableau 5.2. : Services offerts par la AMPRO4

On s'aperçoit que ces services offerts se distinguent considérablement de ceux offerts par la couche transport ISO 8072/73. D'une part, tous les services se font en deux temps : un *request* et une *confirmation*. Aucun des service ne donne lieu à une *indication* auprès de la station paire. D'autre part, ce ne sont pas les mêmes services qui sont offerts. Il y a des services qui sont nécessaires pour l'allocation des *buffer* (ce mécanisme est expliqué par après). La figure 5.8. illustre par un diagramme séquence-temps un déroulement possible d'une connexion (*fig. 5.8. Déroulement possible d'une connexion*).

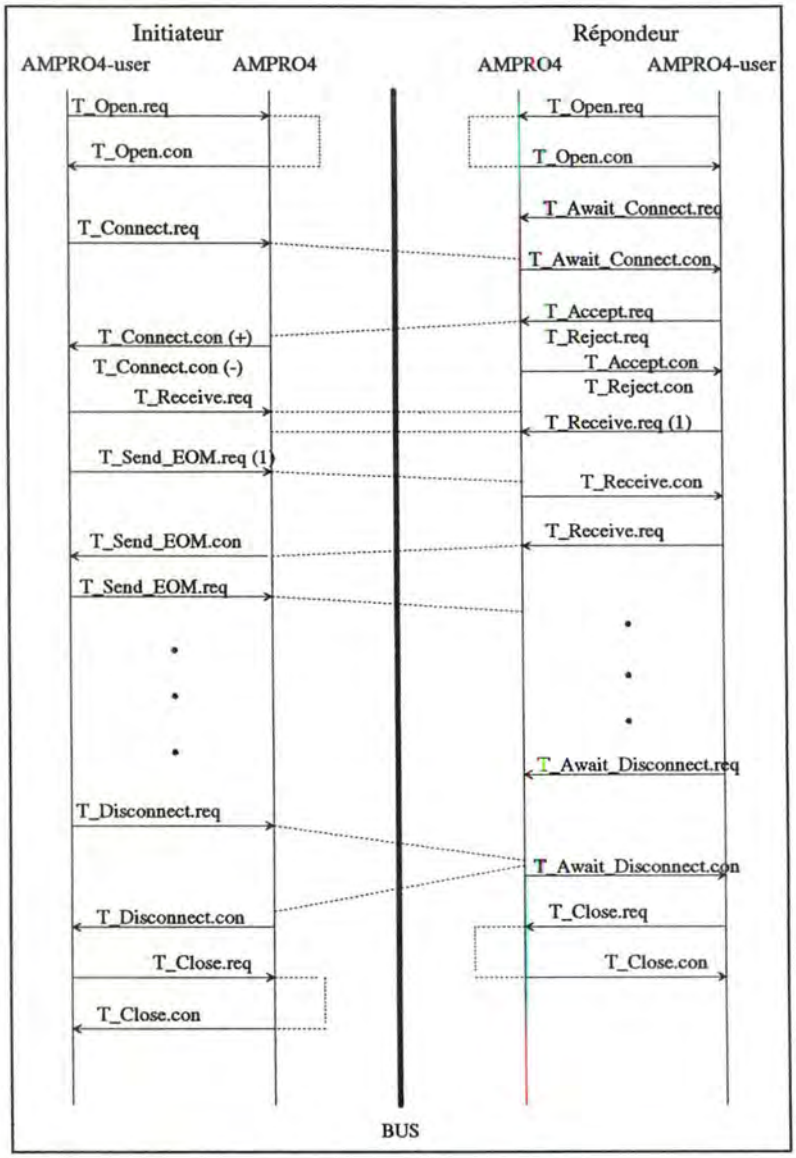


fig. 5.8. Déroulement possible d'une connexion

Avec la AMPRO4, la SPS90 a la possibilité de communiquer facilement avec des stations qui figurent sur un autre réseau. La couche 4 offre là un contrôle d'erreurs et de flux au niveau 4, ce qui n'est possible ni pour Mini-MAP, ni pour Profibus.

LA CONSULTATION D'AUTRES TRAVAUX

Après avoir fait le cahier des charges, une étape intermédiaire, avant de passer à la gestion interne, est l'analyse de l'existant. La lecture d'autres travaux déjà

réalisés s'impose pour repérer des solutions éventuelles. Pour la AMPRO4, le travail concerné est réf. /14/ où l'on peut surtout faire une analyse sur base

- des fonctionnalités
- des services offerts
- de la gestion des ressources
- de la gestion des *buffer*

En fin de compte, nous nous sommes aperçus que, bien que les 2 documents (le cahier des charges et réf. /14/) décrivent une couche transport, les concepts sont pourtant distincts et qu'on ne peut pas reprendre telles quelles les solutions proposées dans le travail de Panzer Bernhard (réf. /14/) pour la AMPRO4.

LA GESTION INTERNE

Le cahier des charges spécifie surtout le protocole et le comportement externe de la couche transport. Mais ce qu'il faut aussi décrire est la manière dont on peut réaliser ce qu'on spécifie. Cette partie de la spécification est beaucoup plus proche d'une implémentation. L'implémentation proprement dite se basera sur la gestion interne.

Voyons quelques idées fondamentales pour la gestion interne telles que la gestion des ressources, la gestion des *buffer* et le passage des informations d'une couche à une autre ainsi que d'un numéro identificateur d'une connexion.

Pour que la couche 4 puisse fonctionner, il lui faut des ressources (*buffer*, place mémoire). Cette place mémoire doit être demandée au système d'exploitation et ne peut être allouée, dans notre cas, que par la AMPRO7. Ni la couche 4, ni la couche 2 ne peuvent allouer elles-mêmes des ressources. La couche 4 reçoit donc nécessairement toutes les ressources indispensables au bon fonctionnement par la couche 7.

C'est alors à la couche 4 de gérer ces ressources, qui circulent entre la couche 4 elle-même et la couche 2, mais aussi de gérer les ressources prêtes à retourner au AMPRO4-*user*. Le chemin des ressources est schématiquement représenté à la figure 5.9.

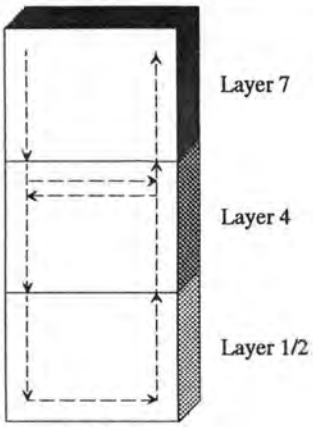


fig. 5.9. Le chemin des ressources.

La gestion efficace et simple de toutes ces ressources est une des tâches principales de la couche AMPRO4. Reprendre toute la suite logique des réflexions faites à ce sujet excéderait considérablement le cadre de ce travail. Contentons-nous alors des choses suivantes. Il est important qu'un seul pointeur puisse passer d'une couche à l'autre. De plus, la couche 4 ne doit pas devoir séparer des informations contenues dans un grand *buffer*.

Avec chaque demande de service, un *T_request_bloc* avec d'autres *buffer* est envoyé à la couche 4. Le *T_request_bloc* retourne à la couche 7 avec la confirmation du service.

Les *T_Send_EOM* sont gérés séparément et on travaille toujours avec deux *buffer* situés dans la couche 2 afin de limiter la retransmission due à des erreurs de *no_ressource*.

Pour recevoir quoi que ce soit, on a besoin de ressources. Ces ressources sont, par exemple, données par les demandes de service *T_Await_Connect.req* et *T_Await_Disconnect.req*.

Avec chaque demande de service, la couche AMPRO7 envoie aussi les *buffer* et les informations nécessaires à la couche 4. La couche 4 doit gérer les différents *buffer*. Dans la spécification de la AMPRO4, cette gestion est une partie essentielle du travail. Il faut que la gestion soit rapide, simple et ne prenne pas trop de place. Plusieurs solutions distinctes sont imaginables. Mais plutôt que de les expliquer toutes, nous expliquerons celle qui a été retenue dans la spécification.

On exige que lors d'une demande de service une adresse suffise pour mettre à la disposition de la couche 4 toutes les ressources et informations nécessaires. Chacune des ressources est aussi accessible par adresse (par un pointeur). Ces

ressources et informations sont donc rattachées à un bloc dont l'adresse est passée avec la demande de service.

Nous faisons la distinction entre deux types de blocs. Premièrement, on a le bloc passé de la couche 7 à la couche 4 et appelé *T_req_bloc*. Deuxièmement, on a le block passé de la couche 4 à la couche 2 et nommé *L_req_bloc*. Chacun des *req_blocs* contient toute l'information nécessaire pour la couche inférieure à laquelle il est transmis.

Les différents blocs doivent être gérés dans et par la couche 4. Pour ce faire, nous utilisons une liste pour les *T_req_blocs* et une autre liste pour les *L_req_blocks* pour chaque connexion. Les demandes de service *T_send_EOM* sont gérées séparément afin de satisfaire entre autre la suite logique des messages.

Une description schématique du *T_req_bloc* est à la figure 5.10.. Nous ne reprenons dans le bloc que les pointeurs vers les *buffer*. Les autres informations ne nous intéressent pas ici.

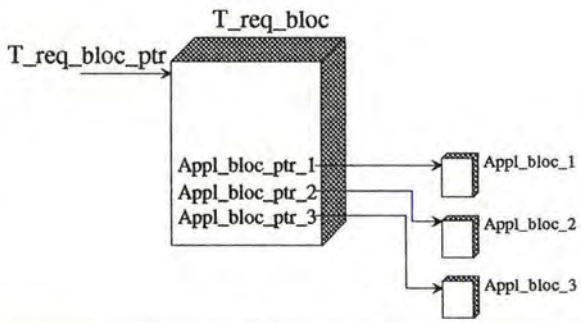


fig. 5.10. Description schématique du T_req_blocs

Pour le *L_req_bloc*, une distinction supplémentaire est établie. On parle, d'une part, d'un *receive buffer* qui correspond à un *L_req_bloc* pour recevoir quelque chose et a par conséquent une seule ressource qui lui est attachée (la SPS90 n'envoie pas de IR). D'autre part, on parle d'un *send buffer* qui lui sert à envoyer un message. A ce *buffer* est nécessairement rattaché un premier *buffer* avec le message à envoyer. En plus, un deuxième *buffer* est attaché au *send buffer* pour recevoir une IR éventuelle. Le troisième type, l'*application bloc*, sert à l'échange d'informations entre la couche 4 et la couche 2 d'une même station et n'a pas de *buffer* qui lui est rattaché. La figure 5.11. ne reprend que les pointeurs qui nous intéressent en ce moment.

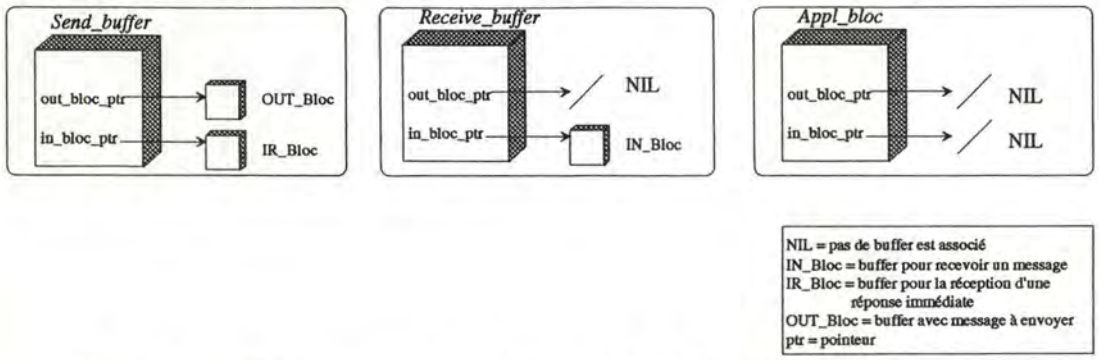


fig. 5.11. Description schématique des différents L_req_blocs

Le nombre de connexions parallèles est limité à 8 et la couche 4 ne peut gérer que 40 *T_req_block* en même temps. En outre, le nombre de *T_Send_EOM* gérables en même temps par la couche 4 est limité à 10. Théoriquement, ce nombre pourrait être plus élevé, mais il répond parfaitement aux besoins.

La figure 5.12. montre schématiquement la gestion des *T_req_blocs*. Pour chaque connexion, il existe une liste des *T_req_blocs*.

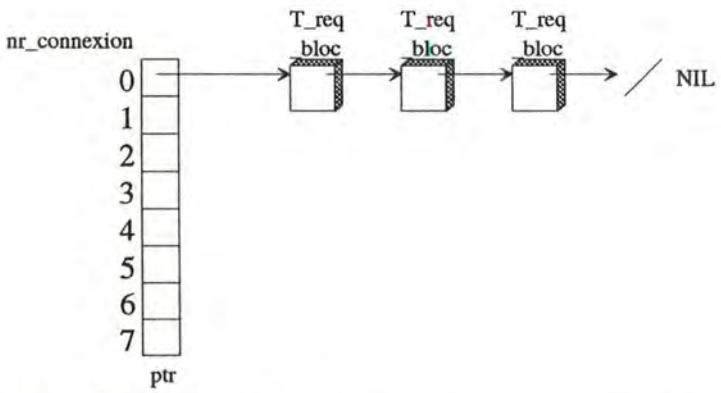


fig. 5.12. Schéma de gestion des T_req_blocs

Pour gérer efficacement les demandes de service *T_send_EOM*, on pour chaque connexion une liste des *T_send_EOM*. On utilise une liste gérée par le principe FIFO pour respecter l'ordre des demandes de service.

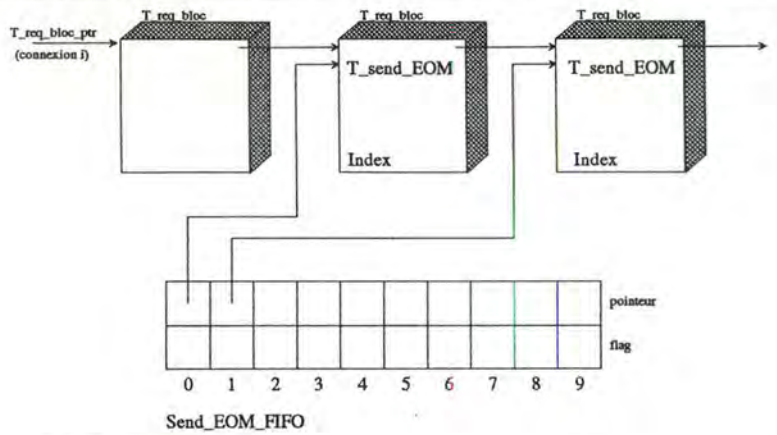


fig. 5.13. Schéma de gestion des T_Send_EOM

Chacune des 8 connexions possibles est parfaitement identifiée par une *connection identification* (CN_ID). A l'ouverture d'une connexion, cette CN_ID est attribuée à la connexion. Lors de la déconnexion, on libère à nouveau ce numéro. Cette CN_ID est utilisée dans la suite par le AMPRO4-*user* pour identifier la connexion lors d'une demande de service à la AMPRO4.

C'est à la AMPRO4 d'attribuer et donc de gérer les CN_ID. Cette gestion peut s'effectuer facilement par un tableau d'une colonne et d'un chaînage entre les champs (les CN_ID) libres. La variable Next_CN_ID contient l'index du champ (CN_ID) suivant. Cette variable est attribuée à une nouvelle connexion. Et la deuxième dans la liste devient le Next_CN_ID. La figure 5.14. montre une situation possible.

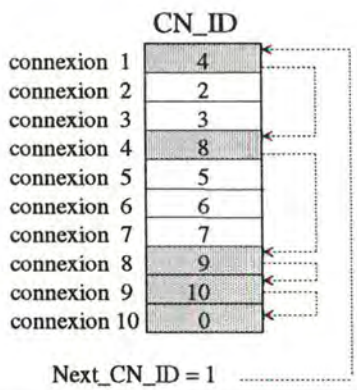


fig. 5.14. Gestion de la CN_ID

Lors de la déconnexion, la CN_ID est libérée et est mise en tête de la chaîne des CN_ID possibles. La CN_ID est propre à une station. Pour une même

connexion, la CN_ID n'est pas nécessairement la même auprès des deux stations connectées.

LES DIAGRAMMES SÉQUENCE-TEMPS

L'étape suivante dans le développement de la couche transport est l'établissement de diagrammes séquence-temps. Ces diagrammes reprennent une situation possible de suites chronologiques de services. Pour la clarté et pour être complet (vollständigkeitshalber) on y trouve aussi les services de la couche 2. L'annexe C donne une liste des services offerts par la couche 2. Plus de détails se trouvent dans la spécification de AMPRO2 (réf. /19/).

En outre, les diagrammes ne reprennent que les cas positifs c.-à.-d. des cas sans erreurs, car les cas d'erreurs sont presque innombrables. Pour simplifier davantage la représentation, nous ne montrerons qu'une seule connexion.

Les diagrammes séquence-temps permettent de visualiser le bon déroulement d'une connexion à partir de son ouverture jusqu'à sa fermeture. Les diagrammes servent entre autre à l'établissement des diagrammes SDL et à l'implémentation et ils sont vraiment utiles comme référence et comme visualisation du déroulement.

Il est vrai qu'un prototype de diagramme séquence-temps se trouve dans le cahier des charges, mais il s'agit d'un diagramme plutôt général qui ne reprend ni les services de la couche 2, ni les PDU qu'il faut envoyer.

Les trois schémas suivants montrent un déroulement possible d'une connexion de l'ouverture de connexion (*fig. 5.15. Connect*) à la déconnexion (*fig. 5.17. Disconnect*) en passant par l'envoi de messages (*fig. 5.16. Send*).

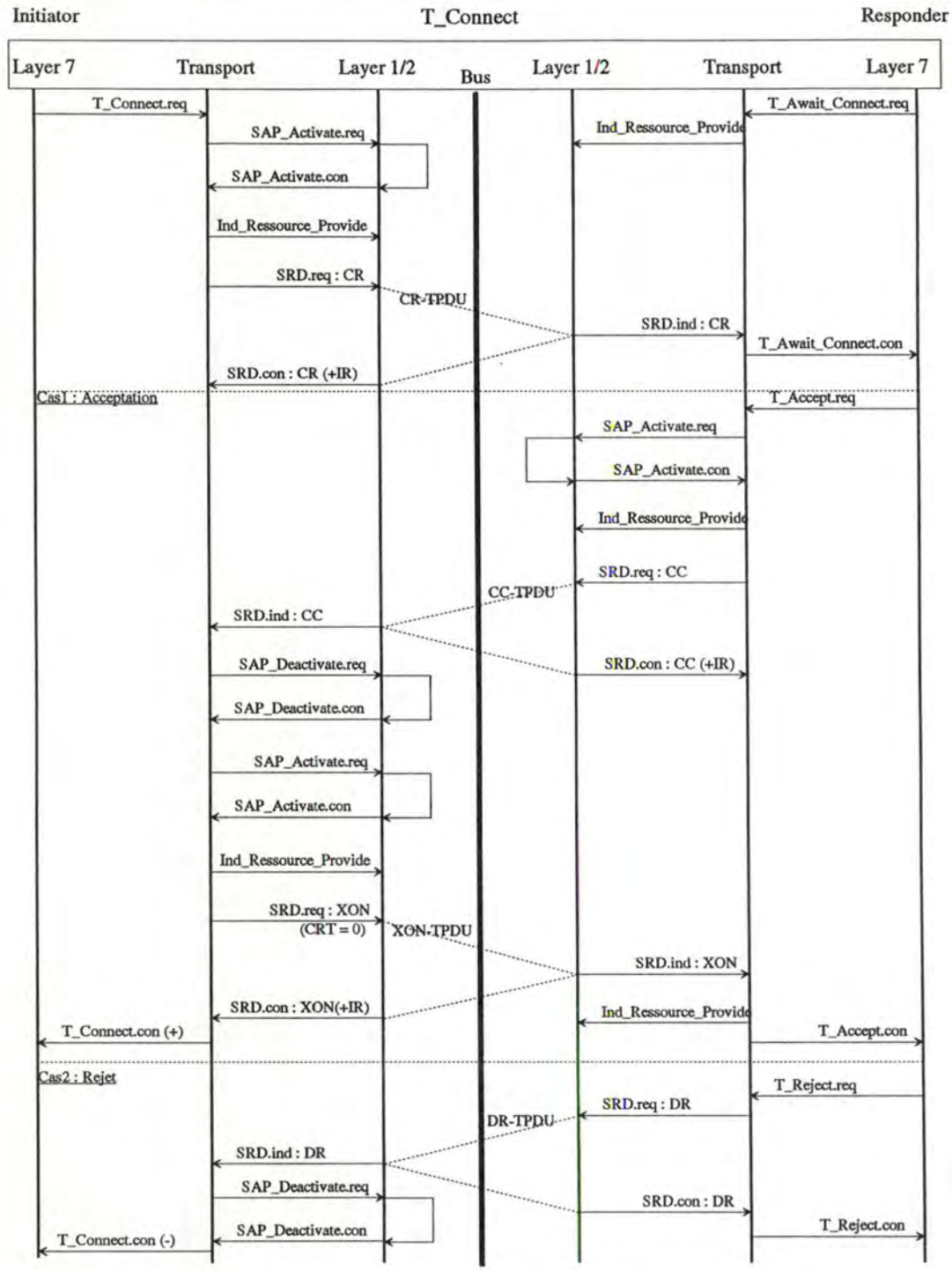


fig. 5.15. Connect

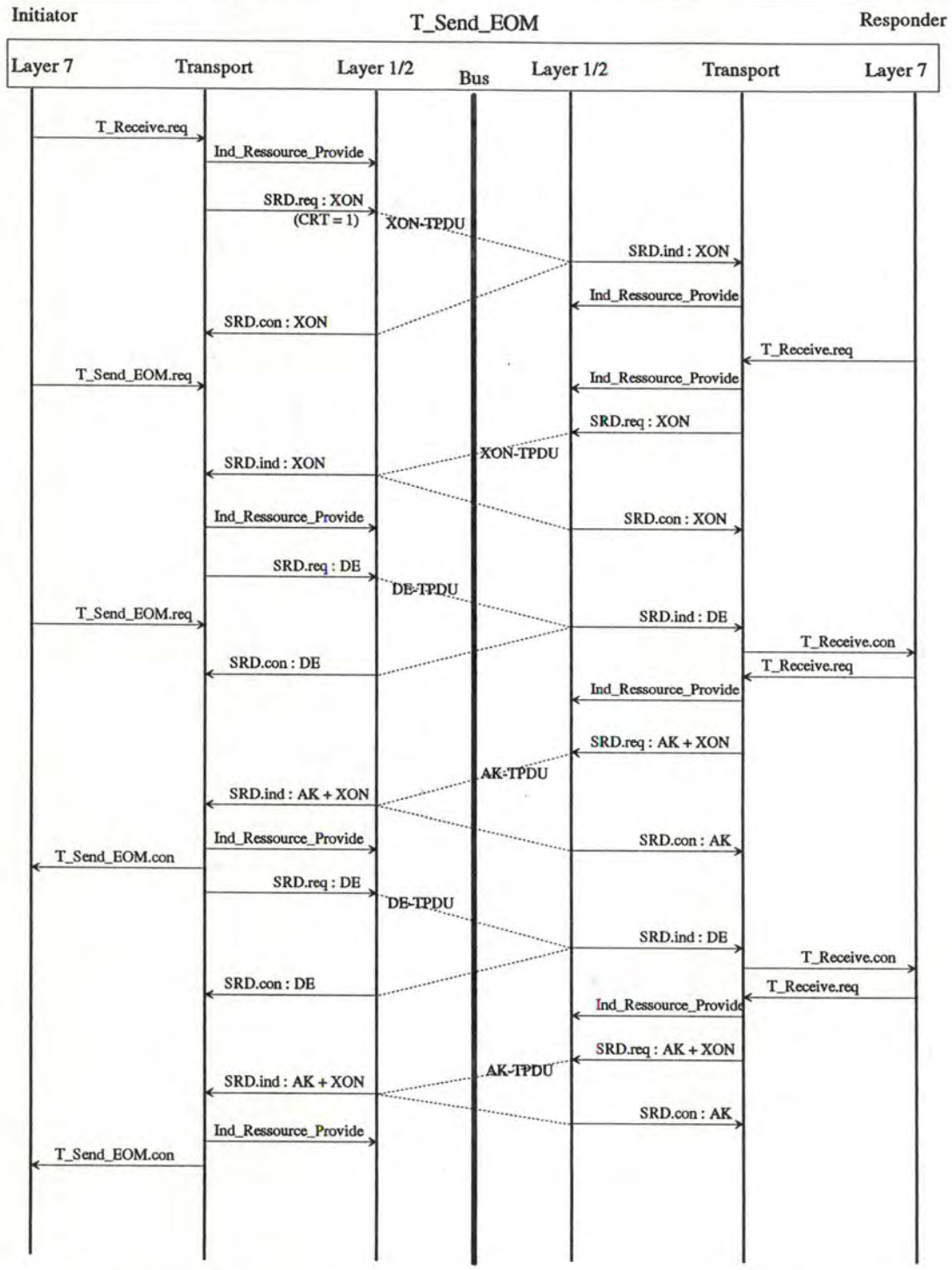

Initiator
T_Send_EOM
Responder
Layer 7
Transport
Layer 1/2
Bus
Layer 1/2
Transport
Layer 7
T_Receive.req
Ind_Ressource_Provide
SRD.req : XON
(CRT = 1)
XON-TPDU
SRD.ind : XON
Ind_Ressource_Provide
SRD.con : XON
T_Receive.req
T_Send_EOM.req
Ind_Ressource_Provide
SRD.req : XON
XON-TPDU
SRD.ind : XON
SRD.con : XON
Ind_Ressource_Provide
SRD.req : DE
DE-TPDU
T_Send_EOM.req
SRD.ind : DE
T_Receive.con
SRD.con : DE
T_Receive.req
Ind_Ressource_Provide
SRD.req : AK + XON
AK-TPDU
SRD.ind : AK + XON
Ind_Ressource_Provide
SRD.con : AK
T_Send_EOM.con
SRD.req : DE
DE-TPDU
SRD.ind : DE
T_Receive.con
SRD.con : DE
T_Receive.req
Ind_Ressource_Provide
SRD.req : AK + XON
AK-TPDU
SRD.ind : AK + XON
SRD.con : AK
Ind_Ressource_Provide
T_Send_EOM.con

fig. 5.16. Send

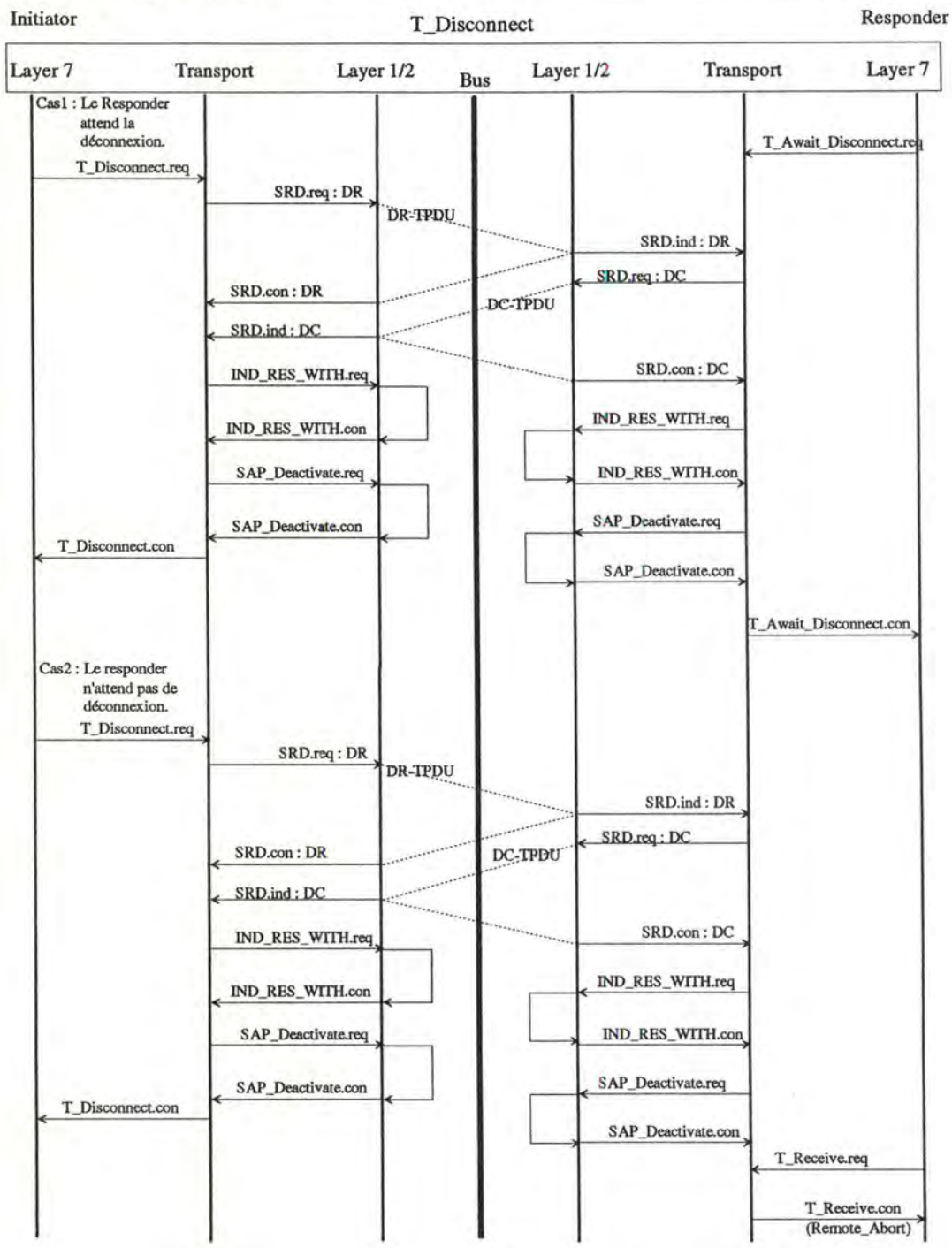


fig. 5.17. Disconnect

Le premier diagramme reprend la phase d'ouverture de connexion (*Connect*). Au cours de cette phase, la SPS90 n'est que le répondeur et non l'initiateur. Elle peut, soit accepter l'ouverture de connexion (*Accept*), soit la refuser (*Reject*).

Pour l'acceptation, l'envoi d'un XON avec *Credit*=0 indique à la station paire que le SAP (Service Access Point) est bien configuré et qu'un échange de messages

sans problèmes est possible. Lorsque la connexion est ouverte, les deux stations sont prêtes à recevoir des données et ont un *Credit* = 1. Pour signaler ce premier *Credit*, la station envoie un XON avec *Credit* = 1 tout seul. Par la suite, le XON ne sera plus jamais envoyé tel quel.

Pour l'envoi et la réception de données, la situation est la suivante : la couche 7 peut faire plusieurs demandes de service Send_EOM à la fois. C'est à la couche 4 de gérer ces demandes. Mais on ne peut demander qu'un *Receive* à la fois. Un nouveau *Receive* ne peut être demandé qu'après la confirmation du précédent.

Le nouveau T_Receive.req provoque aussi l'envoi d'un acquittement (ACK avec XON et *Credit* = 1). Dans le cas où aucun *T_receive* ne suit la dernière confirmation, on n'enverrait plus d'acquittement et le dernier message ne serait pas complètement confirmé (acquitté). Pour cette raison, on prévoit d'envoyer un ACK avec *T_Disconnect* et *T_Await_Disconnect* pour autant que le dernier message reçu n'ait pas encore été confirmé.

Le diagramme de la déconnexion prévoit enfin deux cas : un premier cas qui reprend la séquence où le répondeur attend une déconnexion et est mis à disposition des ressources par *T_Await_Disconnect*; dans le deuxième cas, la déconnexion n'est pas prévue par le répondeur. La couche 4 prévient alors la couche 7 par la confirmation de la prochaine demande de service (p.ex. *T_Receive*) en donnant la raison (*Remote_Abort*)

Il n'est absolument pas dit que le développement de ces diagrammes vienne après la gestion interne, mais il faut ici adopter une démarche séquentielle.

Dans la suite, pour l'ouverture de la connexion, nous ne regarderons plus que le côté répondeur puisque la SPS90 est passive lors de l'ouverture de la connexion.

LES DIAGRAMMES ETATS-TRANSITIONS

Chaque connexion au niveau de la couche 4 est à tout moment dans un état précis. L'élaboration d'un diagramme reprenant l'ensemble des états possibles d'une connexion et l'ensemble des transitions entre ces états aidera ensuite dans le développement des diagrammes SDL.

Plusieurs diagrammes différents ont été développés avant qu'on ne retienne celui qui est représenté à la figure 5.18.. Ce diagramme diffère un peu de l'idée de base qui veut qu'on indique l'événement déclencheur de la transition sur les arcs. Mais la méthode de représentation et le choix des états faciliteront l'élaboration des diagrammes SDL et l'implémentation. C'est ce qu'on espérait en tout cas.

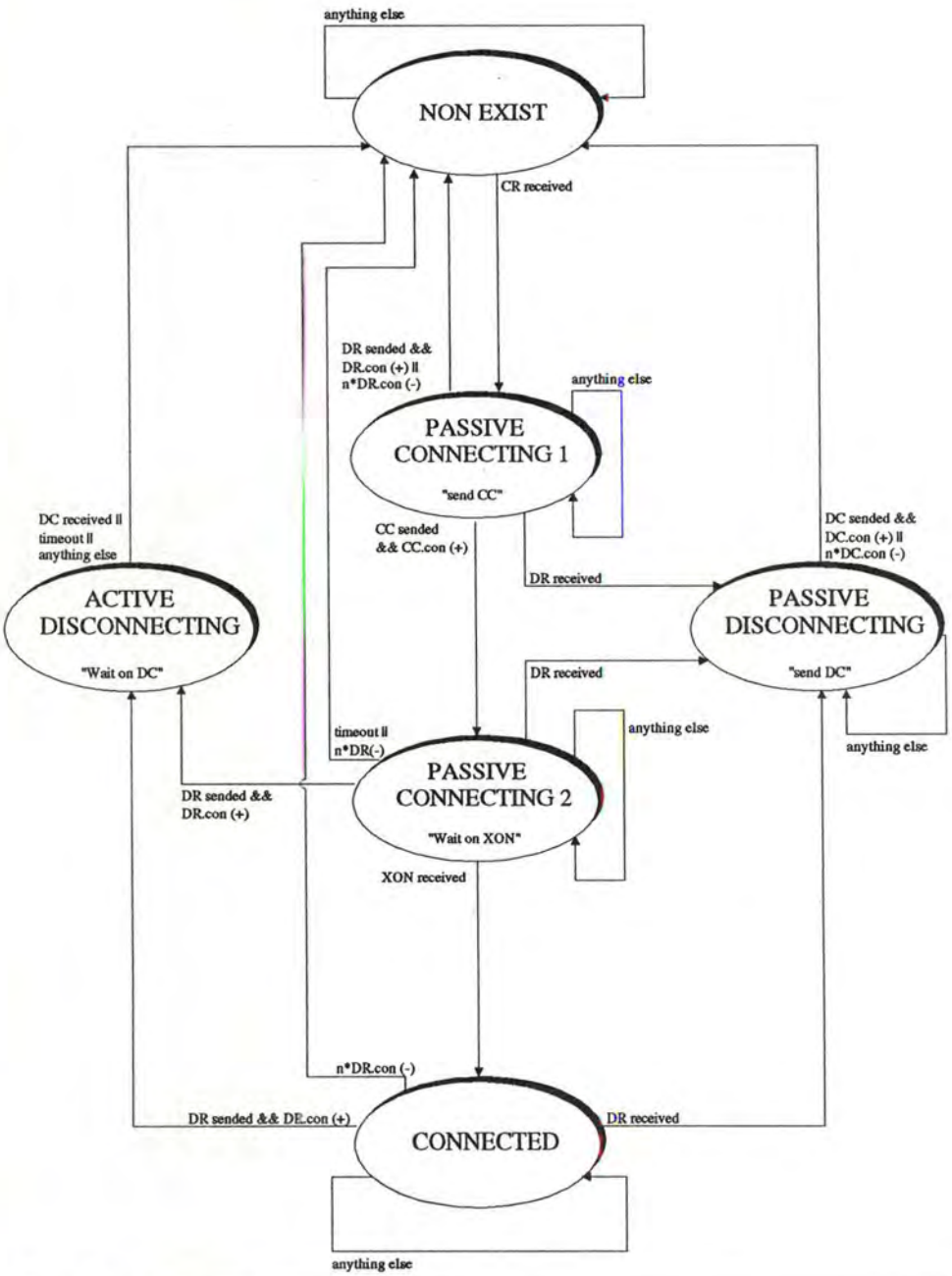


fig. 5.18. Diagramme états-transitions

Le diagramme est formé de 6 états : *NON EXIST*, *Passive Connecting 1*, *Passive Connecting 2*, *Connected*, *Active Disconnecting* et *Passive Disconnecting*. Nous n'avons pas dû ajouter des états de connexion active puisque la SPS90 est passive lors de l'ouverture de la connexion et un état *Active Connecting* n'aurait pas de sens.

Sur les arcs du diagramme figurent les conditions qui doivent être satisfaites pour changer l'état. Les deux barres (||) correspondent à l'*ou logique* et && équivaut à l'*et logique*. Le nombre n correspond au nombre de fois qu'on répète le message en cas d'erreur peu grave. Ce nombre est fixé au lancement de la station.

LES DIAGRAMMES SDL[33]

SDL (*Specification & Description Language*) est une méthode qui permet de rester très flexible et de travailler de manière très structurée, indépendante du système d'implémentation.

L'outil SDL permet de repérer des erreurs avant l'implémentation. SDL permet la génération (au moins partielle) automatique de code et constitue un outil de test et de spécification très performant.

SDL est une méthode utilisée en télécommunication, pour les protocoles de communication, pour des applications en temps réels (ex. : dans l'automatisation) et pour tout autre système basé sur l'échange de messages.

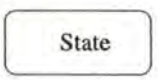
SDL a été conçu pour simplifier la description de processus parallèles ou asynchrones. Les processus échangent des messages et sont en général indépendant les uns des autres. Les idées qui sont à la base du modèle de la méthode SDL sont reprises ci-après :

- Un système est divisé en objets.
- Le comportement dynamique de ces objets est décrit par le processus SDL.
- Les objets ont des états définis et peuvent recevoir des messages.
- Les messages enclenchent un processus après lequel l'objet retourne dans un état défini (automate fini élargi).
- Le processus déclenché peut lors de son exécution envoyer des messages à des processus SDL.

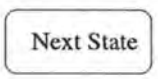
33 référence /15/

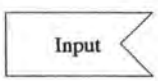
- Les messages peuvent contenir des données.
- La communication est en général asynchrone.
- Un système peut être divisé en sous-systèmes.

La méthode SDL a sa propre grammaire. Les différents éléments de la grammaire sont repris ci-après avec une brève explication.

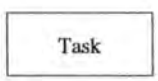


Désigne l'état du processus : Le processus attend l'arrivée d'un message. Tous les états d'un processus ont un nom différent.

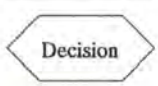




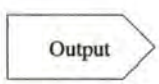
Un processus attend ce message lorsqu'il se trouve dans un état précis.



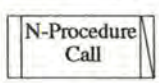
Représente une action à accomplir, par exemple un calcul.



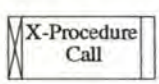
Représente une multiplication des chemins possibles en fonction de valeurs locales. Chaque chemin partant d'un symbole de décision est appelé résultat de la décision. Le résultat est marqué sur la ligne de résultat. Chaque chemin aboutit en une séquence non vide d'actions ou en un symbole Next State ou Stop.



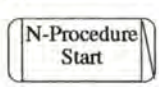
Représente l'envoi d'un message à un autre processus ou au système.



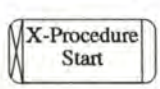
Appel d'une procédure de type N. Ce symbole est mis à la place d'une action ou d'une suite d'actions. Le retour de la procédure de type N est suivi par la continuation des symboles après N-Procedure-Call.



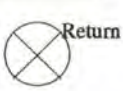
Appel d'une procédure de type X. Ce symbole peut être là où la suite d'actions donne obligatoirement sur un changement d'état ou l'arrêt.



Début d'une procédure de type N. Représente l'entête de la procédure et est au début de la définition de la procédure.



Début d'une procédure de type X. Représente l'entête de la procédure et est au début de la définition de la procédure. Une procédure X se termine obligatoirement par un changement d'état.



Le symbole 'RETURN' représente la fin d'une procédure de type N.

Symbol qui représente un in- ou un out-connector. Il est utilisé pour permettre une structuration plus aisée de grands diagrammes avec beaucoup de branches.

Symbole qui sert à ajouter un commentaire dans le diagramme

Pour montrer une utilisation de cette méthode SDL, nous allons l'illustrer par le cas du développement de la couche 4 AMPRO4. Il est évident que l'ensemble des diagrammes réalisés excède considérablement le cadre de ce travail. D'autre part,

comme nous n'avons pas suivi en détail toute l'élaboration de la AMPRO4, cela prendrait un temps énorme d'en comprendre toute la logique. Pour ces raisons nous avons d'abord décidé de reproduire les diagrammes SDL en annexe. Ensuite, nous nous sommes limités à ne reproduire qu'une simplification des diagrammes réellement utilisés. Ainsi nous faisons, par exemple, comme s'il n'y avait qu'une seule connexion à la fois. De plus, nous avons simplifié toute la gestion des erreurs, partie difficile si on veut être rapide et économe en place mémoire.

Pour donner malgré tout une idée du fonctionnement des diagrammes, nous simulons à la main (à l'annexe D) un déroulement possible d'une connexion. Nous abordons d'abord le refus d'une demande de connexion et ensuite son acceptation.

Pour établir les diagrammes SDL, nous nous sommes surtout orientés vers une implémentation. Cette manière de procéder est plus réaliste et plus claire que celle qui se limite strictement aux diagrammes états-transitions.

Ainsi se termine la partie de spécification. Il est évident que ce que nous venons de présenter précédemment ne ressemble que peu à une spécification, mais ce n'était pas le but. Nous espérons avoir bien présenté les différentes étapes possibles d'une spécification. Reprendre une spécification complète d'une couche 4 demanderait trop de place.

En plus, les points les plus importants et les plus marquants ont été mentionnés. Passons brièvement à la partie implémentation.

5.3.3. L'implémentation

Après avoir spécifié la couche 4, l'étape suivante est l'implémentation. Il est dommage que nous n'ayons pas pu faire l'implémentation de la couche 4, mais une telle implémentation est un processus assez long.

La programmation de la couche 4 se fait en général dans le langage 'C'. Après la phase de programmation s'impose une phase de test avec un émulateur.

Mais même si nous avions implémenté la couche 4, cela n'aurait pas servi à grand chose puisqu'il faut aussi la tester. Par conséquent, il faudrait aussi avoir les couches 1, 2 et 7, un système d'exploitation, etc.. Tout cela n'était pas impossible à réaliser, mais comme on ne peut pas développer tout en détail dans le cadre d'un travail de fin d'études, nous avons dû faire un certain choix.

Par conséquent, l'implémentation ne sera pas traitée dans ce travail.

6. Conclusion

Le mémoire fait le point sur un domaine rarement traité dans le cadre du cours d'informatique des Universités. Il s'agit de l'automatisation d'un processus de fabrication. Le potentiel de communication nécessité dans ce domaine est souvent sous-estimé. Nous avons présenté dans ce travail différents standards de communication adaptés aux différents niveaux de la hiérarchie de communication. Le cadre assez limité de notre travail ne nous a permis que d'aborder les idées principales. Des explications plus en détail et peut-être aussi plus claires exigent plus de place. Une suite de ce travail pourrait être en conséquence une explication plus détaillée (et illustrée par plus d'exemples) des différentes parties et surtout de la partie concernant MMS qui reste la référence en matière de communication industrielle.

La communication industrielle est d'une grande ampleur et concerne beaucoup d'applications industrielles. Il est bien intéressant de vouloir simplifier la communication en adoptant des standards de communication. Mais lorsqu'on veut optimiser la communication, les standards ne sont pas bien adaptés et on commence à s'éloigner du standard.

Par ailleurs, il faut aussi se poser la question si cela vaut la peine de développer une ligne de communication. Il ne faut surtout pas voir la communication comme une fin en soi, mais comme un moyen qui permet d'avoir de nouvelles perspectives. Un automate programmable est là pour contrôler. La communication n'est qu'un plus qu'on offre et ne vient qu'en second lieu.

Tout ceci ne veut pas du tout dire que les standards ne servent à rien. Nous voulons seulement remarquer que souvent, en pratique, les standards sont peu utilisables/utilisés tels quels. Le but qu'on s'est fixé pour OSI est important, mais beaucoup de cas exigent une spécialisation et une adaptation des standards. On pourrait donc croire que l'OSI est du *wishfull thinking*. Il existe toujours des solutions qui s'éloignent plus ou moins du standards communément adopté. Mais un premier pas est fait avec ces standards et s'ils sont bien faits et communément acceptés comme MMS, tous les autres standards et développements y font référence et essayent de s'y adapter.

7. Liste des abréviations

AA	Association Agent
ACK	Acknowledge
ACSE	Association Control Service Element
AE	Association Entity
ALI	Application Layer Interface
AMOS	Advanced Multitasking Operating System
AP	Automation Protocol
APDU	ApplicationProtocol Data Unit
API	Application Programming Interface
APM	AP-Monitor (Protokolabwickler des Sinec AP)
AS	Automatisierungssystem (système d'automatisation)
ASN.1	Abstract Syntax Notation 1
BD	Base de Données
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CAQ	Computer Aided Quality
CC	Connect Confirm (protocole de transport)
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CIM	Computer Integrated Manufacturing
CLNS	Connection Less Network Service
CLTS	Connection Less Transport Service
CNC	Computerized Numeric Control
COTS	Connection Oriented Transport Service
CPU	Central Processing Unit
CR	Connect Request (protocole de transport)
CRT	Credit (protocole de transport)
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSRD	Cyclic Send and Receive Data
DC	Disconnect Confirm (protocole de transport)
DE	Data EOM (protocole de transport)
DEC	Digital Equipement Corporation
DNC	Direct Numerical Control
DPS	Data Processing System
DR	Disconnect Request (protocole de transport)
DT	Data (protocole de transport)
EA	Event Agent
EC	Event Condition
EDI	Electronic Data Interchange
EE	Event Enrollment
EMUG	European MAP User Group
EOM	End Of Message
EPA	Enhanced Performance Architecture

FDDI	Fiber Distributed Data Interchange
FDL	Fieldbus Data Link
FIFO	First In, First Out
FIP	Factory Instrumentation Protocol
FLC	Fieldbus Link Control
FMS	Fieldbus Message Specification
FTAM	File Transfer, Access and Management
FTS	File Transfer Service
GM	General Motors
GPAO	Gestion de production assistée par ordinateur
HP	Hewlett Packard
IBM	International Business Machines
IEEE	Institute of Electrical and Electronic's Engineers
IP	Intelligente Peripherie (appareil périphérique intelligent)
IR	Immediate Response
IS	International Standard
ISA	Instrument Society of America
ISO	International Standardization Organization
ISORM	ISO Reference Model
KBL	Kommunikations Beziehungs Liste (Liste des relations de communication)
LAN	Local Area Network
LAS	List of Active Stations
LLC	Logical Link Control
LLI	Lower Layer Interface
LPDU	data Link Protocol Data Unit
LSAP	Link Service Access Point
LSDU	data Link Service Data Unit
MAC	Medium Access Control
MAN	Metropolitan Area Network
MAP	Manufacturing Automation Protocol
MMFS	Manufacturing Message Format Standard
MMS	Manufacturing Message Specification
NC	Numerical Control
NCC	National Computer Conference
NFS	Network File Service
NPDU	Network Protocol Data Unit
OS	Objektsichtgerät (contrôle par vue et changements rudimentaires)
OS	Operating System
OSI	Open System Interconnection
OV	Objektverzeichnis (directory d'objets)
PDU	Protocol Data Unit
PG	Programmiergerät (Appareil de programmation)
PPDU	Presentation Protocol Data Unit
PROFIBUS	PROcess FIeld BUS

PROWAY	PROcess data highWAY
PSAP	Presentation Service Access Point
QOS	Quality Of Service
ROSE	Remote Operation Service Element
SA	Source Address (Profibus)
SADDR	Source Address (protocole de transport)
SADT	Structured Analysis and Design Technique
SAP	Service Access Point
SCI	Sinec Communication Interface
SDA	Send Data with Acknowledge
SDL	Specification and Description Language
SDN	Send Data with non Acknowledge
SDU	Service Data Unit
SINEC	Siemens Network Architecture for Automation and Engineering
SMF	Standard Message Format
SPDU	Session Protocol Data Unit
SPS	Speicherprogrammierte Steuerung (commande par programme enregistré, automate programmable)
SRD	Send and Receive Data
STF	Sinec Technologische Funktionen (Fonctions Technologiques Sinec)
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TOP	Technical and Office Protocol
TPDU	Transport Protocol Data Unit
TRT	Token Rotation Time
TSAP	Transport Service Access Point
VFD	Virtual Field Device
VMD	Virtual Manufacturing Device
WAN	Wide Area Network
XOFF	Transmit off (protocole de transport)
XON	Transmit on (protocole de transport)

8. Glossaire

Pour l'élaboration de ce glossaire, nous nous sommes explicitement limités aux termes qui nous paraissent intéressant à première vue. Il est évident qu'un tel glossaire ne peut pas être complet. Pour un glossaire plus complet nous conseillons de lire le glossaire des références /3/, /11/ et /12/ desquelles nous nous sommes fortement inspirés.

ACSE — *Association Control Service Element.* Elément de service d'application qui offre les fonctions de base d'ouverture et de fermeture d'une connexion. Peut être utilisé par d'autres éléments de service d'application.

Architecture — Cadre dans lequel les fonctions, interfaces et protocoles d'un système de communication ou de traitement sont spécifiés.

Bande de base — *Baseband.* Transmission d'un signal de données dans sa bande de fréquence d'origine, sans qu'il subisse de modulation.

Bloc — Groupe de bits ou de caractères manipulés comme un tout.

Broadband — L'utilisation d'un câble coaxial pour la transmission de signaux analogues (fréquence radio). Les signaux digitaux passent par un modem et sont transmis sur une de bandes de fréquence du câble.

Broadcast — La transmission simultanée de messages vers l'ensemble des stations connectées.

Carrierband — La même chose qu'un *broadband* à une bande de fréquence.

CIM — *Computer Integrated Manufacturing* (GPAO - Gestion de la production assistée par ordinateur).

Contrôle d'erreurs — Partie de la procédure de liaison permettant la détection et éventuellement la correction d'erreurs de transmission.

Contrôle de flux — Procédure de commande de la cadence de transfert des données entre deux points d'un réseau de données.

Couche application — Couche 7 du modèle de référence ISO.

Couche Liaison de données Couche 2 du modèle de référence ISO.

Couche physique Couche 1 du modèle de référence ISO.

Couche session Couche 5 du modèle de référence ISO.

Couche réseau Couche 3 du modèle de référence ISO.

couche présentation Couche 6 du modèle de référence ISO.

Couche transport Couche 4 du modèle de référence ISO.

Distance de Hamming La distance de Hamming entre deux mots binaires (de même longueur) est le nombre d'éléments binaires de même rang qui diffèrent dans les deux mots.

FTAM *File Transfer, Acces and Management.* Elément de service d'application. Pour permettre une certaine indépendance par rapport à la sauvegarde physique, FTAM utilise le concept de *virtual filestore*.

réf. /12/

Gateway Station permettant d'interconnecter deux ou plusieurs réseaux d'architectures différentes.

Liaison point à point Liaison logique ou physique entre deux participants.

Maître-esclave *Master/slave.* Le maître s'adresse régulièrement aux esclaves.

Mémoire tampon *Buffer.* Mémoire utilisée pour l'enregistrement temporaire de blocs de données, par exemple dans les systèmes correcteurs d'erreurs, en mode différé, ou pour compenser une différence de débit lorsque des informations sont transmises d'un organe à un autre.

Message Suite de caractères, de symboles ou de signaux représentant une information.

Modèle de référence ISO ISO *reference model.* Modèle en 7 couches pour la construction de systèmes ouverts.

modificateur *modifier.* Conditionnement de l'exécution d'un service confirmé quelconque à la réalisation d'un événement ou le contrôle par sémaphore de l'exécution d'un service confirmé quelconque.

Multicast Transmission simultanée de messages vers un groupe de stations connectées.

Multiplexage — Opération réversible consistant à assembler des signaux issus de plusieurs sources distinctes en un seul signal composite destiné à être transmis sur une voie de transmission commune.

PDU — *Protocol Data Unit.* L'unité de données qui est échangée entre deux couches de stations différentes.

Polling — Contrôle d'accès au bus dans lequel un maître (station centrale) s'adresse régulièrement aux autres stations.

Processus — Séquence d'instructions exécutables une à une sans possibilité de simultanéité.

PROFIBUS — *PROcess FIeld BUS.* Réseau de terrain développé par l'industrie et quelques Universités en Allemagne.

Protocole — Ensemble de conventions nécessaires pour faire coopérer des entités généralement distantes, en particulier pour établir et entretenir des échanges d'informations entre ces entités.

Réponse immédiate — *Immediate Response.* Désigne le message qu'une station passive peut envoyer vers une station active lorsque la dernière lui adresse la parole.

Réseau — Ensemble des unités fonctionnelles qui établissent des circuits de données entre terminaux.

ROSE — *Remote Operations Service Element.* Elément de service d'application pour la communication interactive. Une station lance un processus auprès d'une autre station. Cette dernière essaye d'exécuter le processus et remet les résultats à la première station.

SDU — *Service Data Unit.* Unité de données échangée entre deux couches voisines d'une même station.

Station active — Station qui participe activement au token bus. Station qui reçoit le token.

Station passive — Station qui ne participe pas au token bus et qui ne reçoit en conséquence jamais le token.

Système ouvert — Système pour lequel les interfaces sont connues pour que chacun puisse participer au développement.

Taux d'erreurs résiduelles Rapport du nombre des bits, caractères ou blocs incorrectement reçus mais non détectés ou non corrigés par l'équipement de protection contre les erreurs, au nombre total de bits, caractères ou blocs émis.

Temps de réponse Délai de réaction d'un système à un événement. Dans le cas de l'utilisation d'un système de traitement en mode conversationnel, c'est le temps qui s'écoule entre la fin de l'entrée d'un message par l'opérateur et l'apparition du début de la réponse correspondante.

Temps réel Mode de traitement qui permet l'admission des données à un instant quelconque et l'obtention immédiate des résultats.

Token Suite de bits bien définie qui est passée d'une station vers une autre et qui donne, à la station qui le détient, le droit d'envoyer des messages.

Token Bus Un bus sur lequel on a établi un ring logique. Le droit d'accès au bus est réglé par token.

Token Passing Méthode dans laquelle l'accès au bus est réglé par un message particulier qui est passé d'une station à une autre en formant un ring logique.

TOP *Technical and Office Protocol.* Effort de Boeing d'établir une architecture standard pour la communication dite de bureau. L'initiative venait en parallèle au développement de MAP chez General Motors dans les années '80.

Bibliographie

Ouvrages, normes et spécifications

/1/ BENDER Klaus, *PROFIBUS : DER FELDBUS FÜR DIE AUTOMATION*, Hanser, München und Wien, 1990

/2/ BEYSCHLAG Ulf, *OSI in der Anwendungsebene*, Datacom, 1988

/3/ CONRADS D., HALLING H., Serielle Busse : Neue Technologien, Standards, Einsatzgebiete, Technische Akademie Wuppertal, vde-verlag, Berlin, Offenbach, 1987

/4/ Deutsche Norm, *PROFIBUS : DIN 19245, Teil 1*, Beuth, Berlin, April 1991

/5/ Deutsche Norm, *PROFIBUS : DIN 19245, Teil 2*, Beuth, Berlin, Entwurf August 1990

/6/ HENSHALL J., SHAW S., *OSI explained*, Ellis Horwood Limited, 1988

/7/ ISO 8072-1986, *Information processing systems - Open Systems Interconnection - Transport service definition*, first edition, 15.06.1986

/8/ ISO 8073-1986, *Information processing systems - Open Systems Interconnection, Connection oriented transport protocol specification*, first edition, 15.07.1986

/9/ ISO/DIS 9506-1, *Manufacturing Message Specification, Part 1 : Service definition*, 11-1988

/10/ Jürgen SUPPAN-BOROWKA & Thomas SIMON, *MAP - Datenkommunikation in der automatisierten Fertigung*, zweite aktualisierte Auflage, DATACOM, 1986

/11/ KAUFFELS F.-J., *Lokale Netze : Systeme für den Hochleistungs-Informationstransfer*, dritte erheblich erweiterte und aktualisierte Auflage, DATACOM, 1988

/12/ MACCHI C. - GUILBERT J.-F. et 13 co-auteurs, *Téléinformatique*, nouvelle édition, Dunod, 1987

/13/ NUSSBAUMER H., *Téléinformatique*, Tome IV, Presses Polytechniques et Universitaires Romandes, Lausanne, 1991

/14/ PANZER Bernhard, *"Spezifikation einer Transportfunktionalität für Sinec L2", Diplomarbeit ausgeführt an der Technischen Universität München,* Februar 1991

/15/ Siemens AG, *AKL-SDL Kurs : Die SDL Methode*, E TDV-SWT, 9/88, pp.8.1-8.25

/16/ Sinec AP 1.0, *Teil 2 : AP-Technologische Funktionen Teil 2.1, Teil A : Dienstbeschreibung*, 04/89

/17/ Sinec AP 1.0, *Teil 2 : AP-Technologische Funktionen Teil 2.1, Teil A : Dienstbeschreibung, Arbeitsfassung Eventdienste*, 09/89

/18/ Sinec AP 1.0, *Teil 2 : AP-Technologische Funktionen Teil 2.1, Teil A : Dienstbeschreibung*,Teil 2.2, 05/90

/19/ TRETTER B., Schnittstellenbeschreibung AMPRO2, Spezifikation Siemens Aut E 1112B, 1991

Articles

/20/ ERKENS I., SCHÖPF R., "SINEC : Offene Architektur für die industrielle Kommunikation", *Automatisierungstechnische Praxis atp*, vol. 33, 6/1991, pp.313-318

/21/ SCHOLTYSSEK B., "Erstes MAP 3.0 Netz in Europa", *DATACOM*, 11/1991

/22/ SCHWARZ K., "Manufacturing Message Specification (MMS) : Offene Verständigung in verteilten Systemen der industriellen Automatisierung", *Automatisierungstechnische Praxis atp*, vol. 31, 1/1989, pp. 23-29

/23/ ULRICH A., KÖNIG H., "Beschreibung der Anwendungsdienste im PROFIBUS", *msr*, vol. 34, 9/1991, pp. 338-342

Index

Annexes

Annexe A : Les services offerts par MMS

Annexe B: Les services offerts par STF

Annexe C : Les services offerts par AMPRO2

Annexe D : Les diagrammes SDL

Annexe A : Les services offerts par MMS

Liste complète des ensembles fonctionnels et des services offerts par MMS ainsi que la fonction des services.

Gestion générale de l'environnement MMS

Initiate *req/ind/resp/conf*	Etablissement de la communication entre deux utilisateurs de MMS et établissement de l'association.
Conclude *req/ind/resp/conf*	Terminaison ordonnée de l'association et de la communication entre utilisateurs de MMS
Abort *req/ind*	Terminaison abrupte de l'association et de la communication entre utilisateurs de MMS.
Cancel *req/ind/resp/conf*	Annulation d'un service confirmé en cours.
Reject *ind*	Indication d'une erreur de protocole par le fournisseur du service MMS.

Gestion de la VMD

Status *req/ind/resp/conf*	Demande de l'état de la VMD.
UnsolicitedStatus *req/ind*	Signalisation de l'état de la VMD.
GetNameList *req/ind/resp/conf*	Obtention des noms des objets de la VMD.
Identify *req/ind/resp/conf*	Identification de la VMD.
Rename *req/ind/resp/conf*	Changements de noms d'objets de la VMD.
GetCapabilityList *req/ind/resp/conf*	Obtention de la liste des capacités de la VMD

Gestion des domaines

InitiateDownloadSequence *req/ind/resp/conf*	Création d'un domaine par chargement.
DownloadSegment *req/ind/resp/conf*	Chargement d'un segment du domaine.
TerminateDownloadSequence *req/ind/resp/conf*	Fin du chargement du domaine.
InitiateUploadSequence *req/ind/resp/conf*	Lancement de la sauvegarde du domaine.
UploadSegment *req/ind/resp/conf*	Sauvegarde d'un segment du domaine.
TerminateUploadSequence *req/ind/resp/conf*	Fin de la sauvegarde du domaine.
RequestDomainDownload *req/ind/resp/conf*	Lancement par le serveur MMS du chargement par le serveur de fichier.
RequestDomainUpload *req/ind/resp/conf*	Lancement par le serveur MMS de la sauvegarde vers le serveur de fichier.

LoadDomainContent *req/ind/resp/conf*	Lancement du téléchargement.
StoreDomainContent *req/ind/resp/conf*	Lancement de la télésauvegarde.
GetDomainAttributes *req/ind/resp/conf*	Lecture des attributs du domaine.
DeleteDomain *req/ind/resp/conf*	Purge du domaine.

Gestion des instances de programmes

CreateProgramInvocation *req/ind/resp/conf*	Création d'une instance de programme.
DeleteProgramInvocation *req/ind/resp/conf*	Suppression d'une instance de programme.
Start *req/ind/resp/conf*	Lancement d'une instance de programme.
Stop *req/ind/resp/conf*	Suspension d'une instance de programme.
Resume *req/ind/resp/conf*	Reprise d'une instance de programme.
Reset *req/ind/resp/conf*	Réinitialisation d'une instance de programme.
Kill *req/ind/resp/conf*	Destruction d'une instance de programme.
GetProgramInvocationAttributes *req/ind/resp/conf*	Lecture des attributs d'une instance de programme.

Accès aux variables

Read *req/ind/resp/conf*	Lecture par le client de valeurs de variables du serveur.
InformationReport *req/ind*	Notification au client par le serveur de la valeur de variables.
Write *req/ind/resp/conf*	Ecriture par le client de variables du serveur.
DefineNamedVariable *req/ind/resp/conf*	Création d'une variable nommée.
DefineScatteredAccess *req/ind/resp/conf*	Création d'une variable à accès dispersé.
DefineNamedVariableList *req/ind/resp/conf*	Création d'une liste nommée de variables.
DefineNamedType *req/ind/resp/conf*	Création d'un type nommé.
DeleteVariableAccess *req/ind/resp/conf*	Suppression de variables nommées et à accès dispersé.
DeleteNamedVariableList *req/ind/resp/conf*	Suppression d'une liste nommée de variables.
DeleteNamedType *req/ind/resp/conf*	Suppression d'un type nommé.
GetVariableAccessAttributes *req/ind/resp/conf*	Lecture des attributs d'une variable MMS nommée, anonyme, ou à accès dispersé.
GetScatteredAccessAttributes *req/ind/resp/conf*	Lecture des attributs d'une variable à accès dispersé.
GetNamedVariableListAttributes *req/ind/resp/conf*	Lecture des attributs d'une liste nommée de variables.

GetNamedTypeAttributes *req/ind/resp/conf*	Lecture des attributs d'un type nommé.

Gestion des sémaphores

TakeControl *req/ind/resp/conf*	Prise de contrôle d'un sémaphore distant.
RelinquishControl *req/ind/resp/conf*	Libération d'un sémaphore distant.
DefineSemaphore *req/ind/resp/conf*	Création d'un sémaphore banalisé.
DeleteSemaphore *req/ind/resp/conf*	Destruction d'un sémaphore banalisé.
AttachToSemaphore *modificateur*	Contrôle par sémaphore de l'exécution d'un service confirmé quelconque.
ReportSemaphoreStatus *req/ind/resp/conf*	Lecture de l'état global d'un sémaphore.
ReportPoolSemaphoreStatus *req/ind/resp/conf*	Lecture du nom et de l'état des jetons nommés d'un sémaphore étiqueté.
ReportSemaphoreEntryStatus *req/ind/resp/conf*	Lecture de l'état détaillé des rubriques d'un sémaphore.

Communication avec l'opérateur

Input *req/ind/resp/conf*	Lecture d'un message fourni par la station opérateur.
Output *req/ind/resp/conf*	Affichage d'un message sur la station opérateur.

Gestion des événements

EventNotification *req/ind*	Notification par le serveur au client d'une transition d'une condition événementielle.
AcknowledgeEventNotification *req/ind/resp/conf*	Acquittement par le client d'une notification d'événement expédiée par le serveur.
AttachToEventCondition *modificateur*	Conditionnement de l'exécution d'un service confirmé quelconque à la réalisation d'un événement.
AlterEventConditionMonitoring *req/ind/resp/conf*	Modification d'attributs d'une condition événementielle scrutée.
TriggerEvent *req/ind/resp/conf*	Déclenchement d'une condition événementielle déclenchée.
DefineEventCondition *req/ind/resp/conf*	Création d'un condition événementielle.
DeleteEventCondition *req/ind/resp/conf*	Destruction d'une ou de plusieurs conditions événementielles.
GetEventConditionAttributes *req/ind/resp/conf*	Lecture des attributs d'une condition événementielle.
ReportEventConditionStatus *req/ind/resp/conf*	Lecture de l'état d'une condition événementielle.
DefineEventEnrollment *req/ind/resp/conf*	Création d'un enregistrement d'événement
DeleteEventEnrollment *req/ind/resp/conf*	Destruction d'un enregistrement d'événement.

GetEventEnrollmentAttributes *req/ind/resp/conf*	Lecture des attributs d'un ou de plusieurs enregistrements d'événements.
ReportEventEnrollmentStatus *req/ind/resp/conf*	Lecture de l'état d'un enregistrement d'événement.
AlterEventEnrollment *req/ind/resp/conf*	Modification d'attributs d'un enregistrement d'événement.
DefineEventAction *req/ind/resp/conf*	Création d'une action événementielle.
DeleteEventAction *req/ind/resp/conf*	Destruction d'une action événementielle.
GetEventActionAttributes *req/ind/resp/conf*	Lectures des attributs d'une action événementielle.
ReportEventActionStatus *req/ind/resp/conf*	Détermination du nombre d'enregistrements d'événement attachés à une action événementielle.
GetAlarmSummary *req/ind/resp/conf*	Obtention par un client d'un récapitulatif de l'état courant des conditions événementielles scrutés et des enregistrements d'événements notifiés correspondants.
GetAlarmEnrollmentSummary *req/ind/resp/conf*	Obtention par un client d'un récapitulatif de l'état courant des enregistrements d'événement notifiés et des attributs des conditions événementielles scrutées correspondantes.

Gestion de journal

ReadJournal *req/ind/resp/conf*	Lecture d'une ou de plusieurs rubriques de journal éventuellement par des filtres.
WriteJournal *req/ind/resp/conf*	Insertion d'une ou de plusieurs rubriques dans un journal.
InitializeJournal *req/ind/resp/conf*	Suppression d'une partie ou de la totalité des rubriques d'un journal.
ReportJournalStatus *req/ind/resp/conf*	Obtention par un client du nombre de rubriques d'un journal et de la valeur de l'attribut 'destructible par MMS'.
CreateJournal *req/ind/resp/conf*	Création d'un journal vide.
DeleteJournal *req/ind/resp/conf*	Suppression d'un journal

Annexe B : Le nom des services offerts par STF

Les ensembles fonctionnels et les services offerts par STF. Liste des dénominations allemandes et leur correspondant anglais faisant la liaison avec MMS. Nous reprenons d'abord le nom allemand et après son correspondant anglais.

Allgemeine Dienste für virtuelle Geräte	Environment and general management
Status_Virtuelles_Gerät	Status
Status_Virtuelles_Gerät_Melden	UnsolicitedStatus
Namensliste_Abfragen	GetNameList
Identifiziere_Virtuelles_Gerät	Identify
Betriebsmittelliste_Abfragen	GetCapabilityList

Applikationsbeziehungsmanagement	VMD support
Einrichten_Applikationsbeziehung	Initiate
Beenden_Applikationsbeziehung	Conclude
Abbruch_Applikationsbeziehung	Abort

Variablendienste	Variable Access
Lesen	Read
Melden	InformationReport
Schreiben	Write
Variablen_Attribute_Abfragen	GetVariableAccessAttributes

Domaindienste	Domain management
Ladesequenz_Beginn	InitiateDownloadSequence
Lade_Segment	DownloadSegment
Ladesequenz_Ende	TermionateDownloadSequence
Hochladesequenz_Beginn	InitiateUploadSequence
Hochlade_Segment	UploadSegment
Hochladesequenz_Ende	TerminateUploadSequence
Laden_Anfordern	RequestDomainDownload
Hochladen_Anfordern	RequestDomainUpload
Lade_Domain_Inhalt	LoadDomainContent
Speichere_Domain_Inhalt	StoreDomainContent
Lösche_Domain	DeleteDomain
Domain_Attribute_Abfragen	GetDomainAttributes

Programminstanzdienste	Program invocation management
Kreiere_Programminstanz	CreateProgramInvocation
Lösche_Programminstanz	DeleteProgramInvocation
Starten	Start
Stoppen	Stop
Fortsetzen	Resume
Rücksetzen	Reset
Abbruch	Kill
Programm_Instanz_Attribute_Abfragen	GetProgrammInvocationAttributes

Eventdienste (pas encore implémenté)	Event management
Einrichten_Event_Condition	DefineEventCondition
Lösche_Event_Condition	DeleteEventCondition
Abfragen_Event_Condition_Zustand	ReportEventConditionStatus
Ändern_Event_Condition	AlterEventConditionMonitoring
Auslösen_Event	TriggerEvent
Einrichten_Event_Action	DefineEventAction
Lösche_Event_Action	DeleteEventAction
Einrichten_Event_Enrollment	DefineEventEnrollment
Lösche_Event_Enrollment	DeleteEventEnrollment
Ändern_Event_Enrollment	AlterEventEnrollment
Abfragen_Event_Enrollment_Zustand	ReportEventEnrollmentStatus
Event_Melden	EventNotification
Quittierung_Event_Melden	AcknowledgeEventNotification

Semaphoredienste	Semaphore management
Belegen_Semaphore	TakeControl
Freigeben_Semaphore	RelinquishControl
Abragen_Semaphore_Zustand	ReportSemaphoreStatus
Abfragen_Pool_Semaphore_Zustand	ReportPoolSemaphoreStatus
Abfragen_Semaphore_Entry_Zustand	ReportSemaphoreEntryStatus

Journaldienste	Journal management
Einrichten_Journal	CreateJournal
Löschen_Journal	DeleteJournal
Lesen_Journal	ReadJournal
Schreiben_Journal	WriteJournal
Rücksetzen_Journal	InitializeJournal
Abfragen_Journal_Status	ReportJournalStatus

Services non ouverts de Sinec TF (et pas présents dans MMS)

Bytestring_Lesen
Bytestring_Schreiben
Transparenter_Datenaustausch
Zeit_Abfragen
Zeit_Einstellen
Zeit_Übertragen

Annexe C : Les services offerts par AMPRO2

Plus de détails sur les services ainsi que leur paramètres et valeurs possibles se trouvent dans la description de l'interface AMPRO2 (réf. /19/). Nous reprenons ici que les services ainsi qu'une brève description.

FLC_FMA_Reset *req/conf*	Remise à 0 de la couche 2.
SAP_Activate *req/conf*	Permet d'activer et de configurer un *service access point* (SAP).
SAP_Deactivate *req/conf*	Permet de désactiver un *service access point* (SAP).
SAP_Status_Read *req/conf*	Permet de lire l'état d'un *service access point* (SAP) local.
FMA_Ind_Resource_Provide req	Met à la disposition de la couche 2 des ressources pour le management (FMA).
FMA_Ind_Resource_Withdraw *req/conf*	Permet de reprendre toutes les ressources FMA.
MAC_Event *ind*	Permet à la couche 2 d'informer la couche 4 des événements survenus.
Statistic_Activate *req/conf*	Activation d'un compteur statistique.
Statistic_Read *req/conf*	Permet de lire le compteur statistique.
SAP_Lock *req/conf*	Rend la réception de données par ce SAP impossible.
SAP_Unlock *req/conf*	Rend inactive un SAP_Lock précédent.
SDA *req/ind/conf*	Demande de service SDA.
SDN *req/ind/conf*	Demande de service SDN.
SRD *req/ind/conf*	Demande de service SRD.
FDL_Status *req/conf*	Demande l'état de la FDL.
Ind_Resource_Provide req	Met une ind_resource à la disposition de la couche 2.
Ind_Resource_Queue_Provide *req/conf*	Met une liste d'ind_resource à la disposition de la couche 2.
Ind_Resource_Withdraw *req/conf*	Retire toutes les *ind_resource* d'un SAP spécifié.
Reply_Update *req/conf*	Permet, pour un SAP local, d'échanger un vieux response_buffer contre un nouveau en faisant toutes les mises à jour nécessaires.

Annexe D : Les diagrammes SDL

Au chapitre 5, nous avons expliqué et présenté les diagrammes SDL. Maintenant, nous donnons un exemple de diagrammes élaborés lors de la spécification d'une couche transport (AMPRO4). Mais avant de présenter les diagrammes, rappelons d'abord les restrictions faites pour ces diagrammes et faisons quelques remarques les concernant.

Des restrictions ont été imposées par le cahier des charges, mais dans le cas que nous présentons, nous avons pris des restrictions supplémentaires pour simplifier la représentation.

- Nous ne représentons que le chemin positif, c.-à.-d que nous ne considérons pas les cas, où les demandes de services, les indications, les confirmations ou autre chose viennent au moment inopportun.
- Nous ne prenons en considération qu'une seule connexion. En réalité, on pourrait en avoir plusieurs en même temps. Cette restriction ne change pas fondamentalement les diagrammes. La seule chose qui diffère c'est qu'on ne doive pas gérer un *pool* des *T_Await_Connect*; ceci simplifie un peu les diagrammes.
- Pour ne pas avoir des diagrammes qui n'expriment rien, il a bien fallu ajouter certains tests. Si ces tests révèlent une erreur, celle-ci est souvent marquée, mais pas détaillée. La réaction à des erreurs fatales n'est expliquée nulle part.
- L'idée d'une brève présentation des cas négatifs est de montrer au lecteur l'existence de cas d'erreurs.
- Nous avons pris la convention suivante : après un *T_Await_Disconnect* la couche 7 ne fait plus de demande de service à la couche 4. Ceci demande naturellement une couche 7 plus intelligente.
- *T_Close* termine 'officiellement' la connexion. C'est la seule demande de service permise après *T_Disconnect* et *T_Await_Disconnect.*
- La fonction *garbage_collection_ex* (activée par l'input *garbage_collection*) représente un moyen de contrôle de la connexion. S'il n'y a pas eu d'activité sur la connexion depuis la dernière *garbage_collection*, il y a quelque chose qui tourne mal et on va initier une déconnexion.

- Dans tous les cas, selon le cahier des charges, nous essayons toujours de terminer convenablement (on attend ou envoie DC) la connexion pour être sûr que les *buffer* retournent à la couche 7.
- Nous avons essayer de rester à un niveau logique assez élevé et de n'employer des noms de variables que là où c'était nécessaire et utile à la compréhension.

Quelques remarques peuvent encore être formulées à propos des diagrammes qui suivent ci-après.

- L'élaboration de ces diagrammes SDL ne se base pas sur la manière classique qui, en partant du diagramme états-transitions, aboutit à des diagrammes SDL où les états (*states*) sont équivalents aux états du diagramme états-transitions. Nous avons choisi de partir d'un état global nommé AMPRO4 et d'y revenir toujours. Cela nous permet d'être un peu plus près de l'implémentation.
- Le niveau des diagrammes est un niveau peu développé. On n'entre pas vraiment dans les détails. D'y trouver déjà des variables, est une question de facilité. Les variables utilisées permettent de mieux comprendre ce qu'on veut dire.
- Le tableau A1 donne une liste des variables les plus importantes ainsi que leur signification.

state_flag	L'état actuel. Correspond à l'état sur le diagramme états-transitions
cn_id	L'identificateur de la connexion.
last_t_service	Le code indiquant la dernière demande de *t_service*.
loc_lsap	LSAP local.
remote_credit	*Credit* de la *remote station*. Indique quel crédit la station a donné à la *remote station*. Permet ainsi de vérifier si le partenaire peut vraiment m'envoyer des données.
local_credit	*Credit* reçu de l'autre station.
seg_nr	Numéro du segment.

tableau C.1. : Liste des variables les plus importantes

Passons donc à présent aux deux exemples qu'on présentera. Le premier, qui est le plus court des deux montre le cas d'un rejet d'une demande de connexion de la part d'une autre station. Le deuxième exemple, à son tour montre un déroulement complet possible d'une connexion entre deux stations.

cas 1 : rejet d'une demande de connexion

La station concernée est passive lors de l'ouverture d'une connexion. La couche 4 de la station est à l'état AMPRO4 (diagramme 1).

A l'arrivée de la demande de service *T_Open* on passe au diagramme 2 (*t_open_ex*). Là, la couche 4 attribue une *CN_ID* à la connexion, initialise une liste de paramètres et donne une confirmation positive à la couche 7. Ensuite, on retourne au diagramme 1 et l'exécution se poursuit.

La demande de service suivante est *T_Await_Connect* (diagramme 3; *t_await_connect_ex*). On met à jour quelques variables et met des ressources à la disposition de la couche 2. Le déroulement se poursuit par après.

On attend alors l'arrivée d'un *SRD.ind:CR* (diagrammes 8, 9, 13) qui provoque une confirmation positive du *T_Await_Connect*. La couche 4 attend alors l'arrivée soit d'une demande de service *T_Accept* soit d'une demande de service *T_Reject*. Dans notre cas, c'est le *T_Reject* (diagramme 5) qui nous occupe. On met alors à jour des variables et on envoie une *DR-TPDU*.

Après la réception de la confirmation (*SRD.con:DR*; diagramme 25) on envoie une confirmation du *T_Reject*.

La procédure est terminée par la demande de service *T_Close* (*t_close_ex*; diagramme 30) et par la confirmation de ce service.

Pour refaire le chemin, il suffit de suivre la suite des diagrammes montrés en se souvenant qu'un retour par *return* ramène toujours au diagramme 1.

diagramme 1	*AMPRO4*
diagramme 2	*t_open_ex*
diagramme 3	*t_await_connect_ex*
diagramme 8	*flc_fma_ind_conf_ex*
diagramme 9	*flc_fma_ind_to_t_ex*
diagramme 13	*cr_ind_to_t_ex*
diagramme 5	*t_reject_ex*
diagrammes 8, 10, 23, 25	*dr_conf_to_t_ex*
diagramme 30	*t_close_ex*

cas 2 : déroulement complet possible

Le début de ce deuxième cas est le même que pour le premier cas et n'est en conséquence pas repris. Nous reprenons là, où la couche 7 peut soit faire la demande de service *T_Accept*, soit celle du *T_Reject*. Le *T_Reject* déjà traité, nous nous occupons à présent du *T_Accept* (diagramme 4). La couche 4 attend la confirmation, de l'activation du SAP. Lorsque cette confirmation arrive (diagrammes 8, 10, 20) on met des ressources à la disposition de la couche 2 et on envoie une CC-TPDU. On attend la confirmation de la CC-TPDU (diagrammes 8, 10, 23, 24) pour attendre alors l'arrivée d'un XON afin d'être sûr que le partenaire a bien configuré son SAP.

Avec l'arrivée de la *SRD.ind:XON* (diagrammes 8, 9, 17), la couche 4 peut confirmer le *T_Accept* et la connexion est établie. L'état du diagramme états-transitions est CONNECTED. Les deux stations peuvent alors échanger des données.

Pour ce faire, il est d'abord nécessaire d'avoir un *receive buffer* disponible qui procure un crédit à la station paire en envoyant un XON avec CRT=1. Ce XON est confirmé (diagrammes 8, 10, 27). La station est alors prête à recevoir des messages.

Dans notre cas, elle en reçoit par une *SRD.ind:DE* (diagrammes 8, 9, 16). On remet à 0 le *remote credit* (crédit qu'on a signalé à l'autre) et on confirme positivement le *T_Receive*.

C'est alors à la station d'envoyer un message. La couche 7 fait une demande de service *T_Send_EOM* (diagramme 6) et la couche 4 envoie en conséquence une DE-TPDU. La couche 4 attend la confirmation de la DE-TPDU (diagrammes 8, 10, 26). Après, elle reçoit une *ACK.ind* avec *credit* = 1 (diagrammes 8, 9, 18) et elle peut alors confirmer positivement le *T_Send_EOM*.

Maintenant, elle reçoit un *T_Disconnect* (diagramme 12) de la couche 7 et la couche 4 envoie une DR-TPDU. La couche 4 attend alors la *SRD.con:DR* (diagrammes 8, 10, 25) puis une *SRD.ind:DC* (diagrammes 8, 9, 15) ensuite elle retire toutes les ressources. Après la confirmation de la dernière opération (diagramme 22) on enclenche la déactivation du SAP.

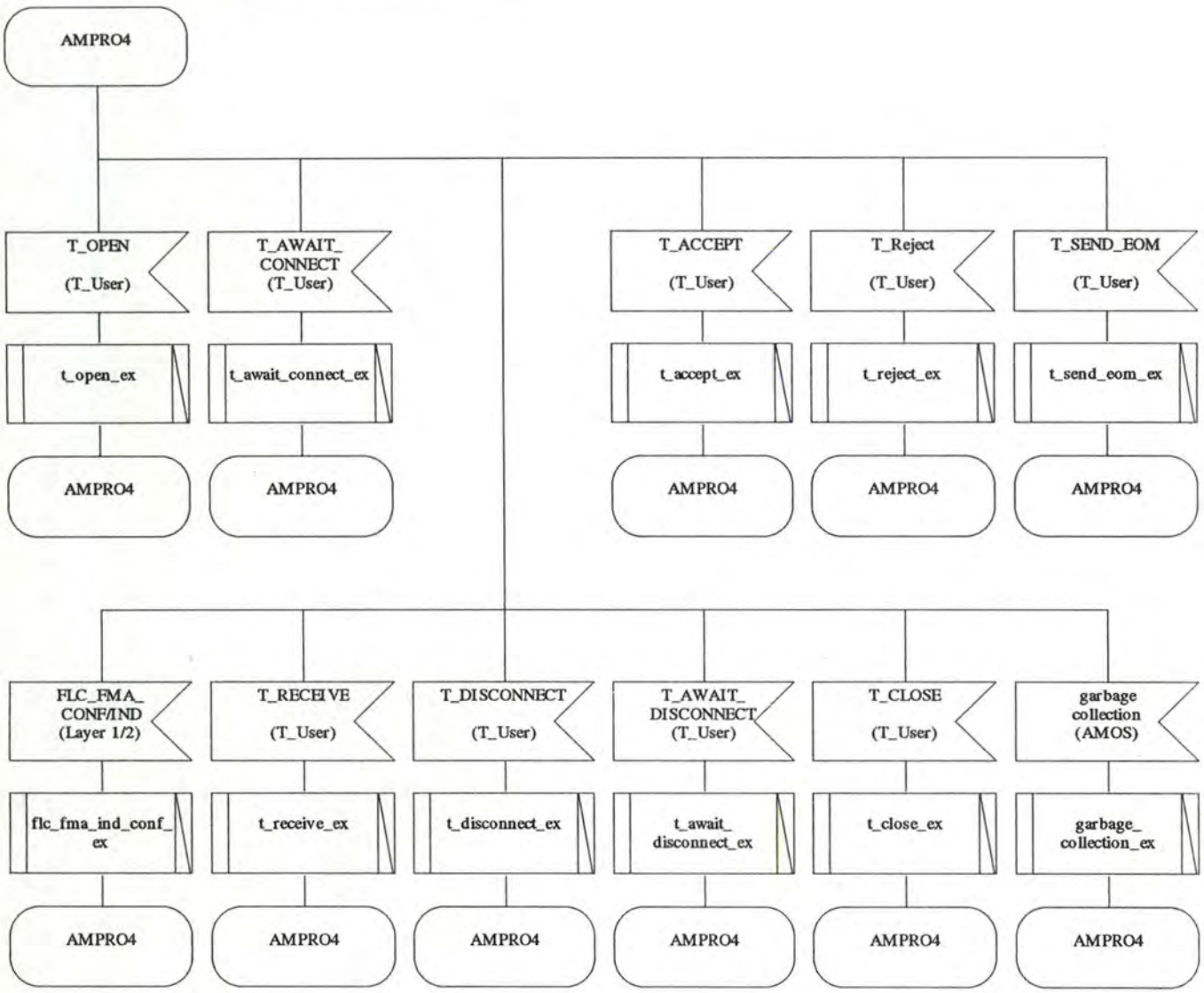
Lorsqu'on reçoit à la couche 4 la confirmation du *SAP_Deactivate* (diagramme 21),on confirme positivement le *T_Disconnect*.

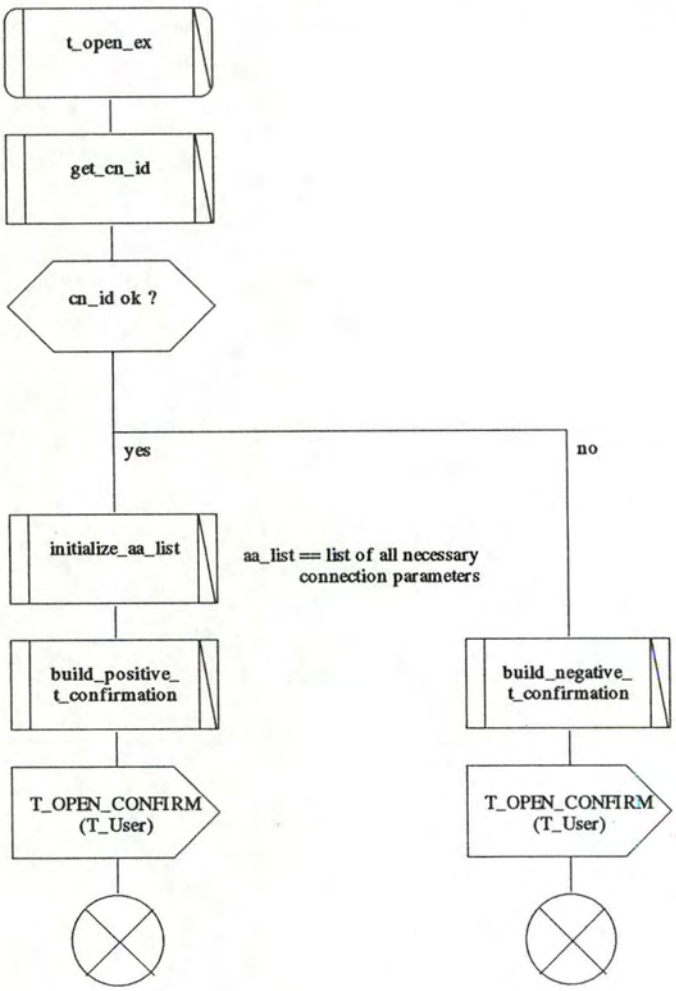
Le reste (*T_Close*) correspond au cas 1 (diagramme 30).

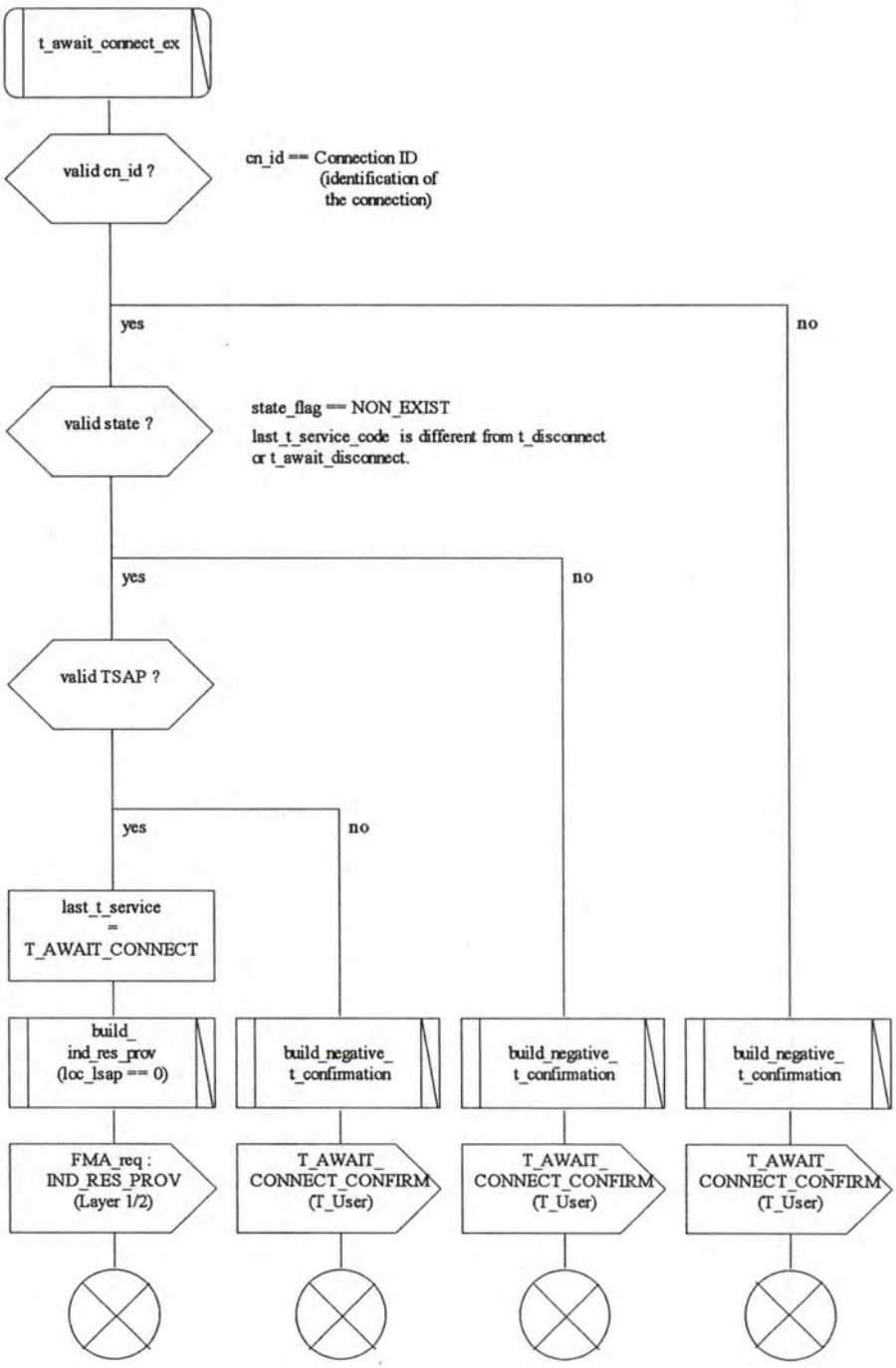
diagramme 1	*AMPRO4*
diagramme 2	*t_open_ex*

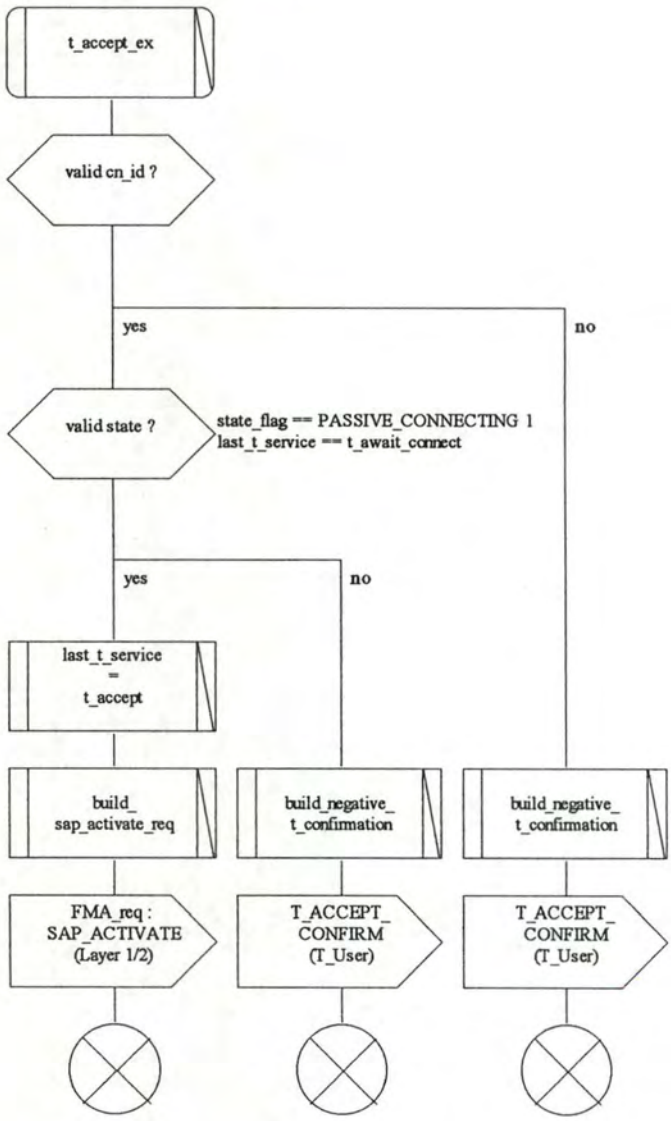
diagramme 3	*t_await_connect_ex*
diagramme 8	*flc_fma_ind_conf_ex*
diagramme 9	*flc_fma_ind_to_t_ex*
diagramme 13	*cr_ind_to_t_ex*
diagramme 4	*t_accept_ex*
diagrammes 8, 9, 20	*sap_activate_conf_to_t_ex*
diagramme 24	*cc_conf_to_t_ex*
diagramme 17	*xon_ind_to_t_ex*
diagramme 7	*t_receive_ex*
diagramme 27	*xon_conf_to_t_ex*
diagramme 16	*de_ind_to_t_ex*
diagramme 6	*t_send_eom_ex*
diagramme 26	*de_conf_to_t_ex*
diagramme 18	*ack_ind_to_t_ex*
diagramme 12	*t_disconnect_ex*
diagramme 25	*dr_conf_to_t_ex*
diagramme 15	*dc_ind_to_t_ex*
diagramme 22	*ind_ress_with_conf_to_t_ex*
diagramme 21	*sap_deactivate_conf_to_t_ex*
diagramme 30	*t_close_ex*

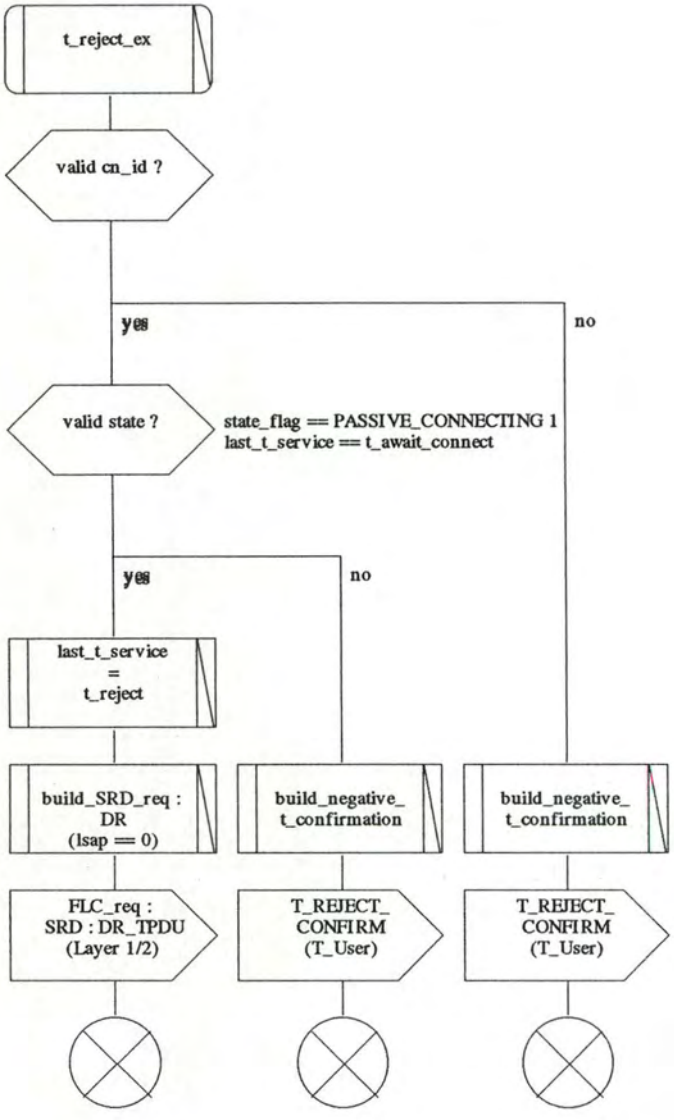
L'utilisation des diagrammes séquence-temps et du diagramme états-transitions aide à lire les diagrammes SDL. La suite documentée n'est qu'une suite possible, mais permet d'illustrer les diagrammes SDL.

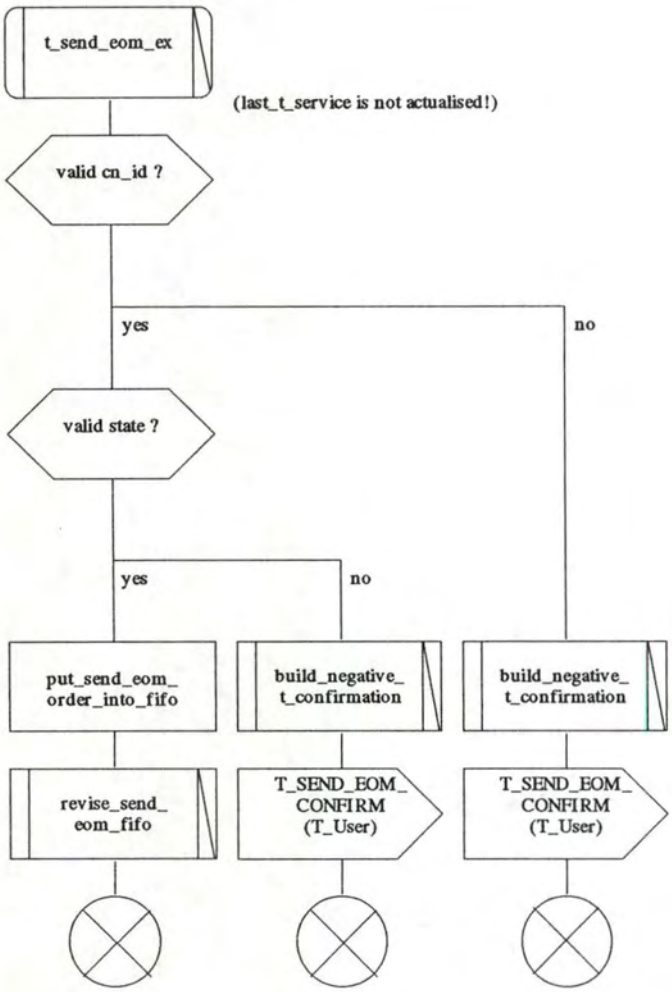

AMPRO4
T_OPEN (T_User)
T_AWAIT_ CONNECT (T_User)
T_ACCEPT (T_User)
T_Reject (T_User)
T_SEND_EOM (T_User)
t_open_ex
t_await_connect_ex
t_accept_ex
t_reject_ex
t_send_eom_ex
AMPRO4
AMPRO4
AMPRO4
AMPRO4
AMPRO4
FLC_FMA_ CONF/IND (Layer 1/2)
T_RECEIVE (T_User)
T_DISCONNECT (T_User)
T_AWAIT_ DISCONNECT (T_User)
T_CLOSE (T_User)
garbage collection (AMOS)
flc_fma_ind_conf_ ex
t_receive_ex
t_disconnect_ex
t_await_ disconnect_ex
t_close_ex
garbage_ collection_ex
AMPRO4
AMPRO4
AMPRO4
AMPRO4
AMPRO4
AMPRO4

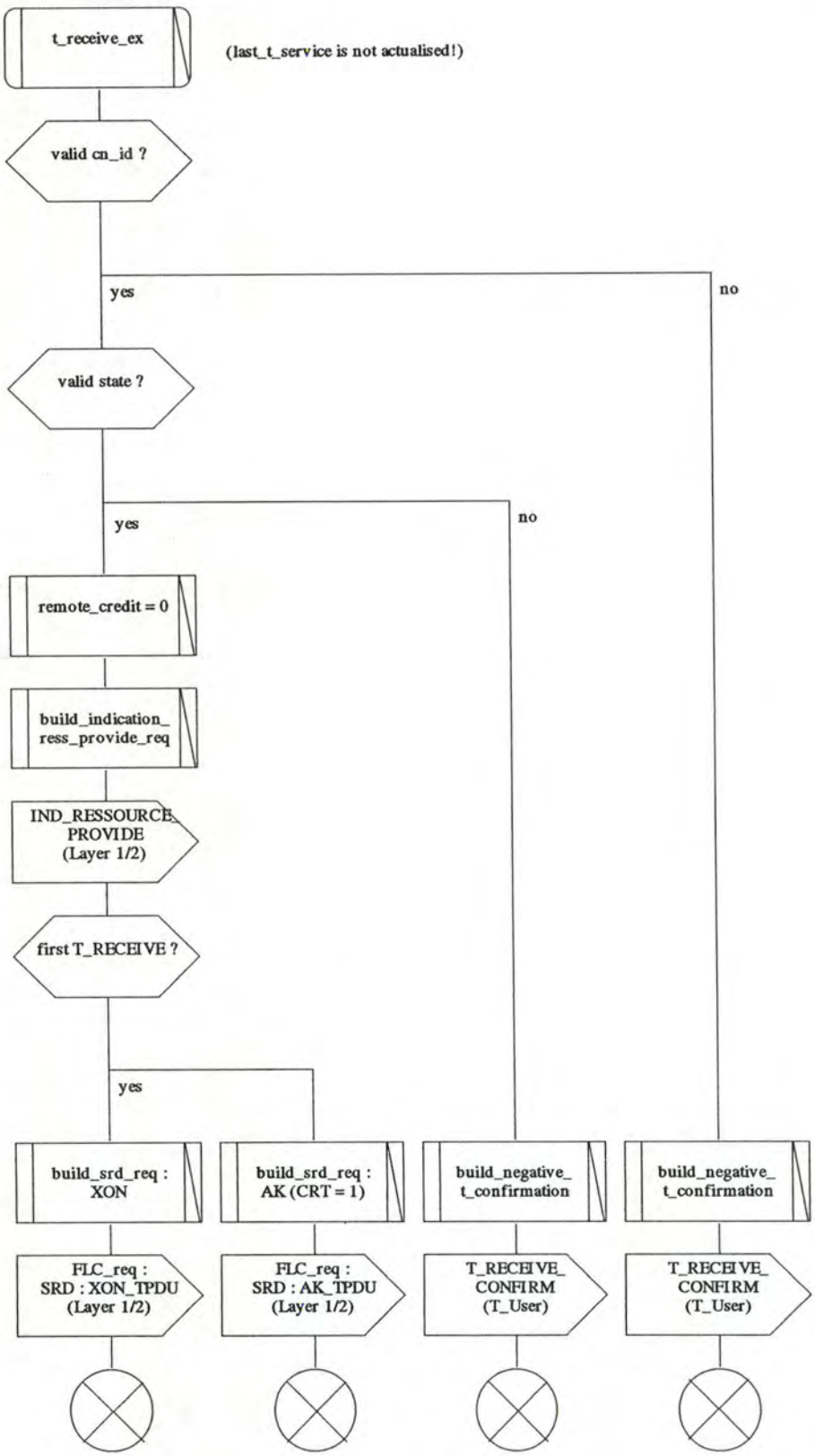

t_open_ex
get_cn_id
cn_id ok ?
yes
no
initialize_aa_list
aa_list == list of all necessary
connection parameters
build_positive_
t_confirmation
build_negative_
t_confirmation
T_OPEN_CONFIRM
(T_User)
T_OPEN_CONFIRM
(T_User)

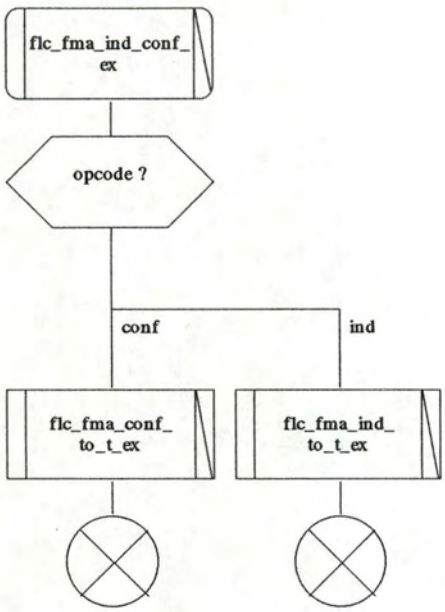

t_await_connect_ex
valid cn_id ?
cn_id == Connection ID
(identification of
the connection)
yes
no
valid state ?
state_flag == NON_EXIST
last_t_service_code is different from t_disconnect
or t_await_disconnect.
yes
no
valid TSAP ?
yes
no
last_t_service
=
T_AWAIT_CONNECT
build_
ind_res_prov
(loc_lsap == 0)
build_negative_
t_confirmation
build_negative_
t_confirmation
build_negative_
t_confirmation
FMA_req :
IND_RES_PROV
(Layer 1/2)
T_AWAIT_
CONNECT_CONFIRM
(T_User)
T_AWAIT_
CONNECT_CONFIRM
(T_User)
T_AWAIT_
CONNECT_CONFIRM
(T_User)

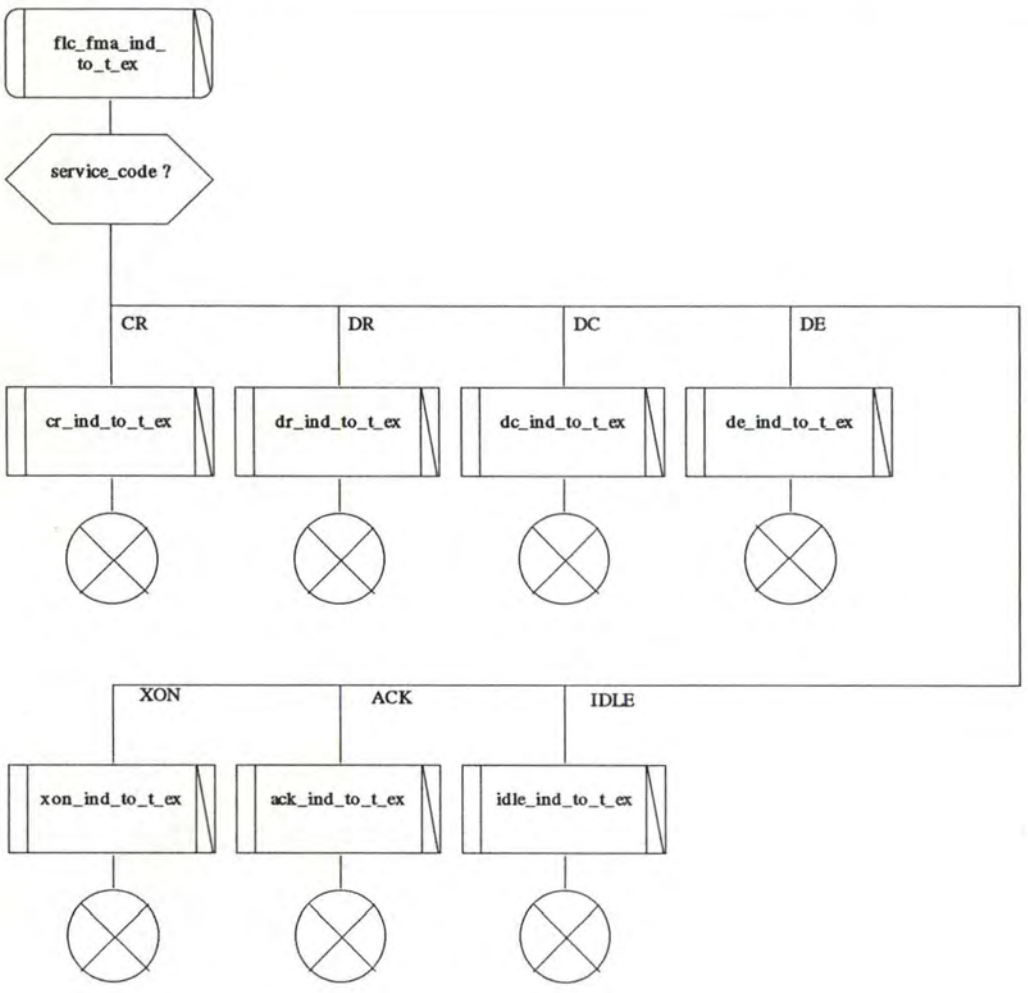

t_accept_ex
valid cn_id ?
yes
no
valid state ?
state_flag == PASSIVE_CONNECTING 1
last_t_service == t_await_connect
yes
no
last_t_service
=
t_accept
build_
sap_activate_req
build_negative_
t_confirmation
build_negative_
t_confirmation
FMA_req :
SAP_ACTIVATE
(Layer 1/2)
T_ACCEPT_
CONFIRM
(T_User)
T_ACCEPT_
CONFIRM
(T_User)

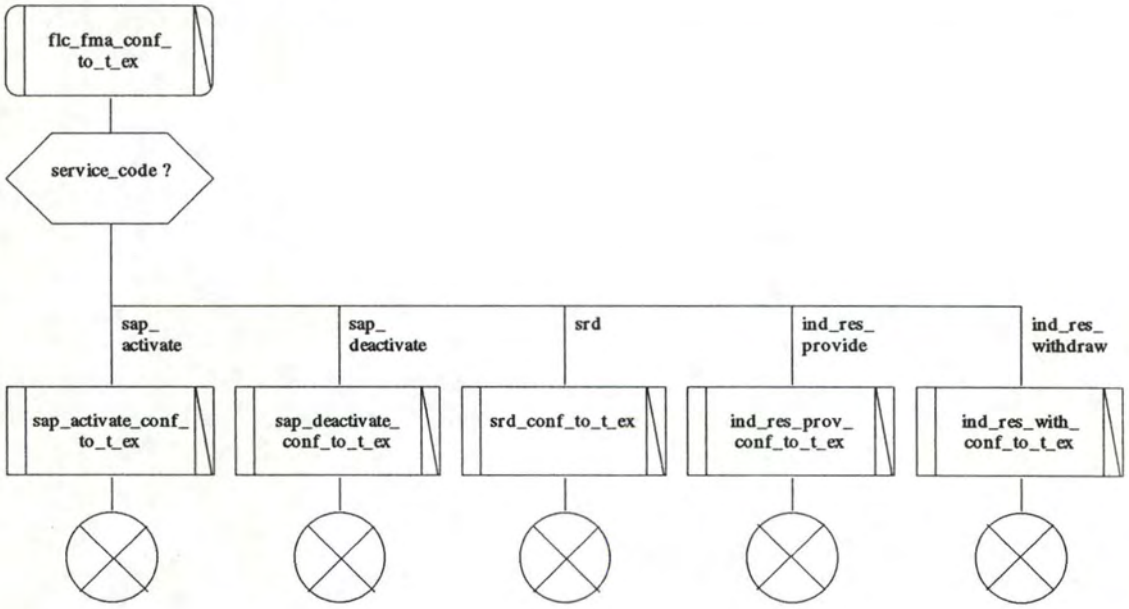

t_reject_ex
valid cn_id ?
yes
no
valid state ?
state_flag == PASSIVE_CONNECTING 1
last_t_service == t_await_connect
yes
no
last_t_service
=
t_reject
build_SRD_req :
DR
(lsap == 0)
build_negative_
t_confirmation
build_negative_
t_confirmation
FLC_req :
SRD : DR_TPDU
(Layer 1/2)
T_REJECT_
CONFIRM
(T_User)
T_REJECT_
CONFIRM
(T_User)

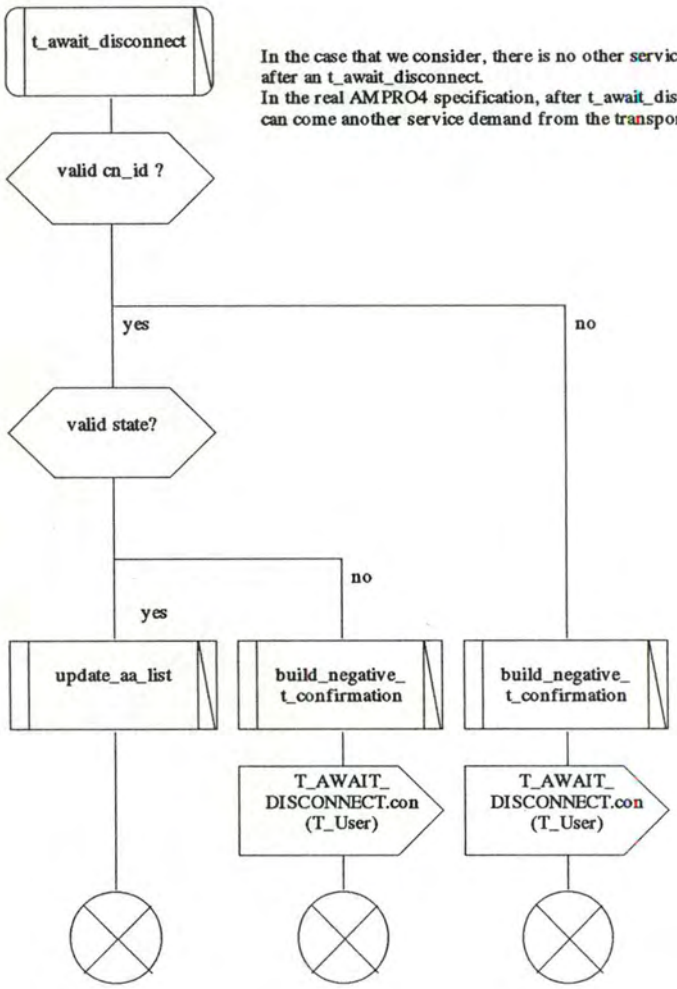

t_send_eom_ex
(last_t_service is not actualised!)
valid cn_id ?
yes
no
valid state ?
yes
no
put_send_eom_
order_into_fifo
build_negative_
t_confirmation
build_negative_
t_confirmation
revise_send_
eom_fifo
T_SEND_EOM_
CONFIRM
(T_User)
T_SEND_EOM_
CONFIRM
(T_User)

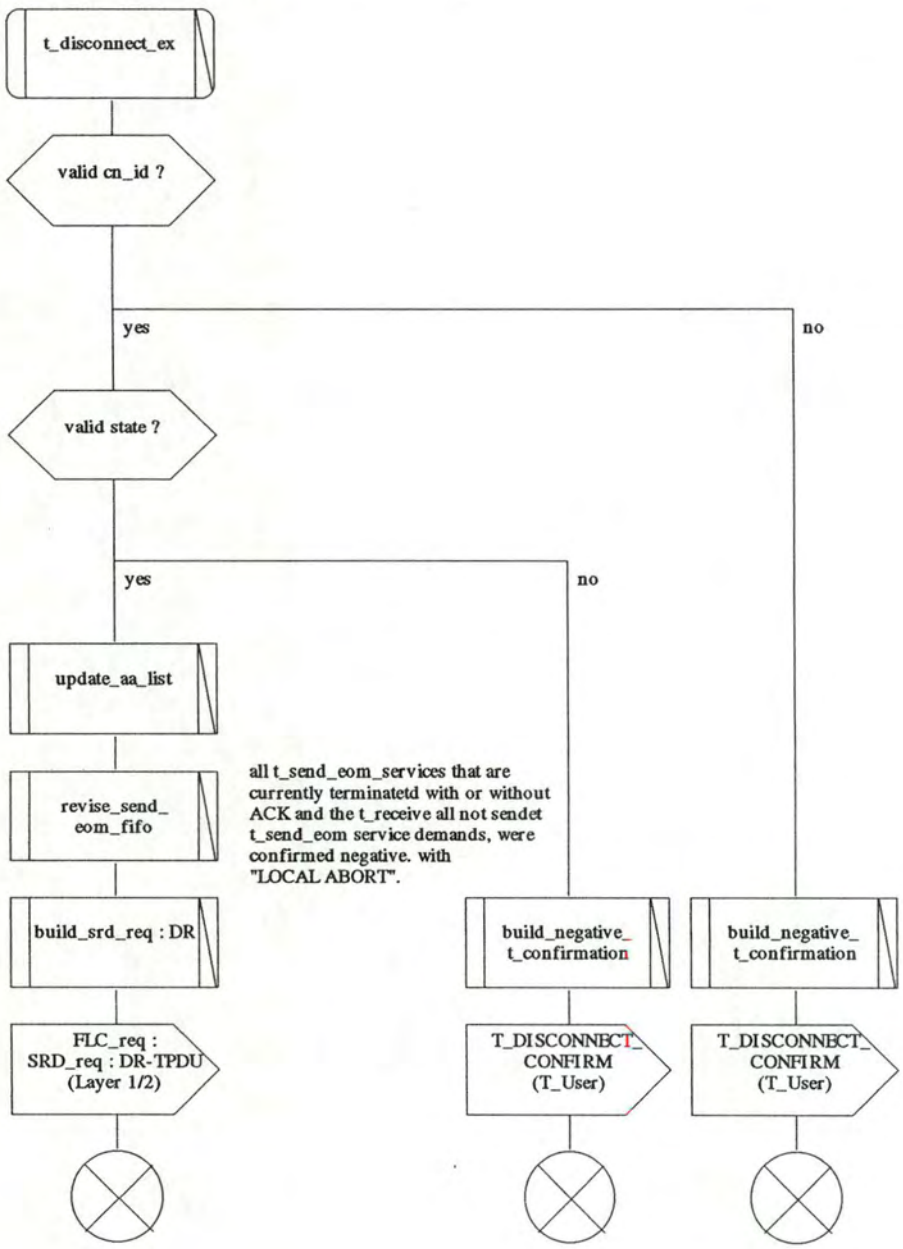

t_receive_ex
(last_t_service is not actualised!)
valid cn_id ?
yes
no
valid state ?
yes
no
remote_credit = 0
build_indication_
ress_provide_req
IND_RESSOURCE_
PROVIDE
(Layer 1/2)
first T_RECEIVE ?
yes
build_srd_req :
XON
build_srd_req :
AK (CRT = 1)
build_negative_
t_confirmation
build_negative_
t_confirmation
FLC_req :
SRD : XON_TPDU
(Layer 1/2)
FLC_req :
SRD : AK_TPDU
(Layer 1/2)
T_RECEIVE_
CONFIRM
(T_User)
T_RECEIVE_
CONFIRM
(T_User)

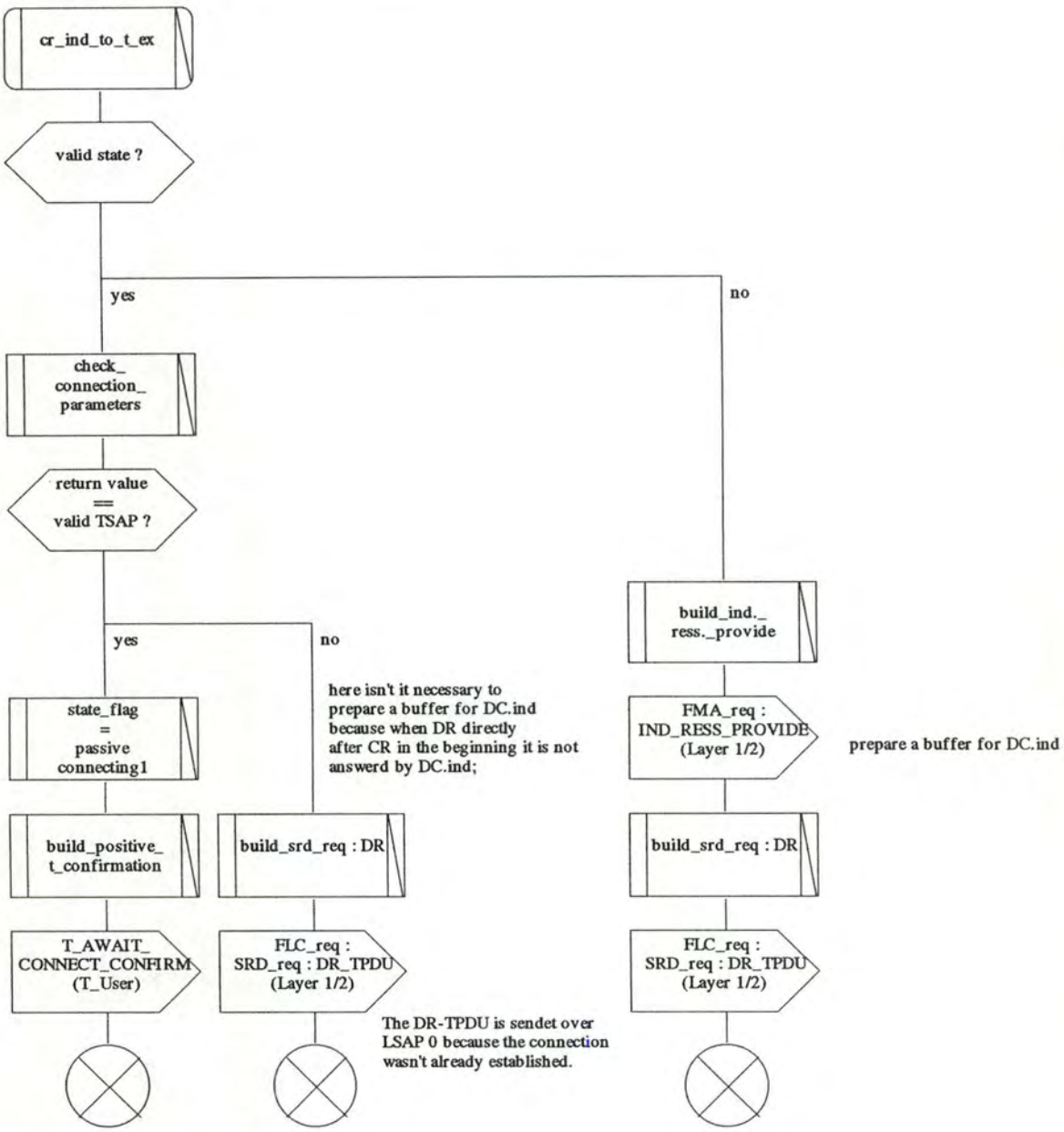

flc_fma_ind_conf_
ex
opcode ?
conf
ind
flc_fma_conf_
to_t_ex
flc_fma_ind_
to_t_ex

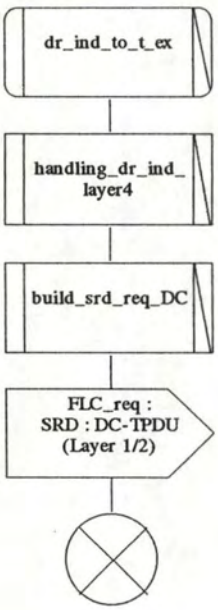

flc_fma_ind_
to_t_ex
service_code ?
CR
DR
DC
DE
cr_ind_to_t_ex
dr_ind_to_t_ex
dc_ind_to_t_ex
de_ind_to_t_ex
XON
ACK
IDLE
xon_ind_to_t_ex
ack_ind_to_t_ex
idle_ind_to_t_ex


flc_fma_conf_
to_t_ex
service_code ?
sap_
activate
sap_
deactivate
srd
ind_res_
provide
ind_res_
withdraw
sap_activate_conf_
to_t_ex
sap_deactivate_
conf_to_t_ex
srd_conf_to_t_ex
ind_res_prov_
conf_to_t_ex
ind_res_with_
conf_to_t_ex

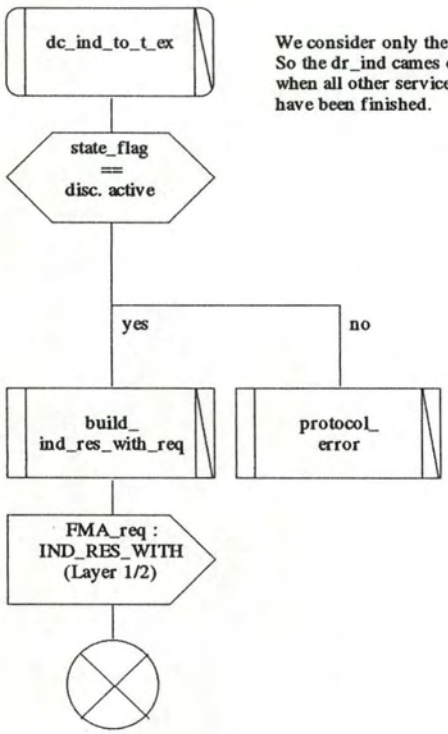

t_await_disconnect
In the case that we consider, there is no other service demand after an t_await_disconnect.
In the real AMPRO4 specification, after t_await_disconnect can come another service demand from the transport user
valid cn_id ?
yes
no
valid state?
no
yes
update_aa_list
build_negative_ t_confirmation
build_negative_ t_confirmation
T_AWAIT_ DISCONNECT.con (T_User)
T_AWAIT_ DISCONNECT.con (T_User)

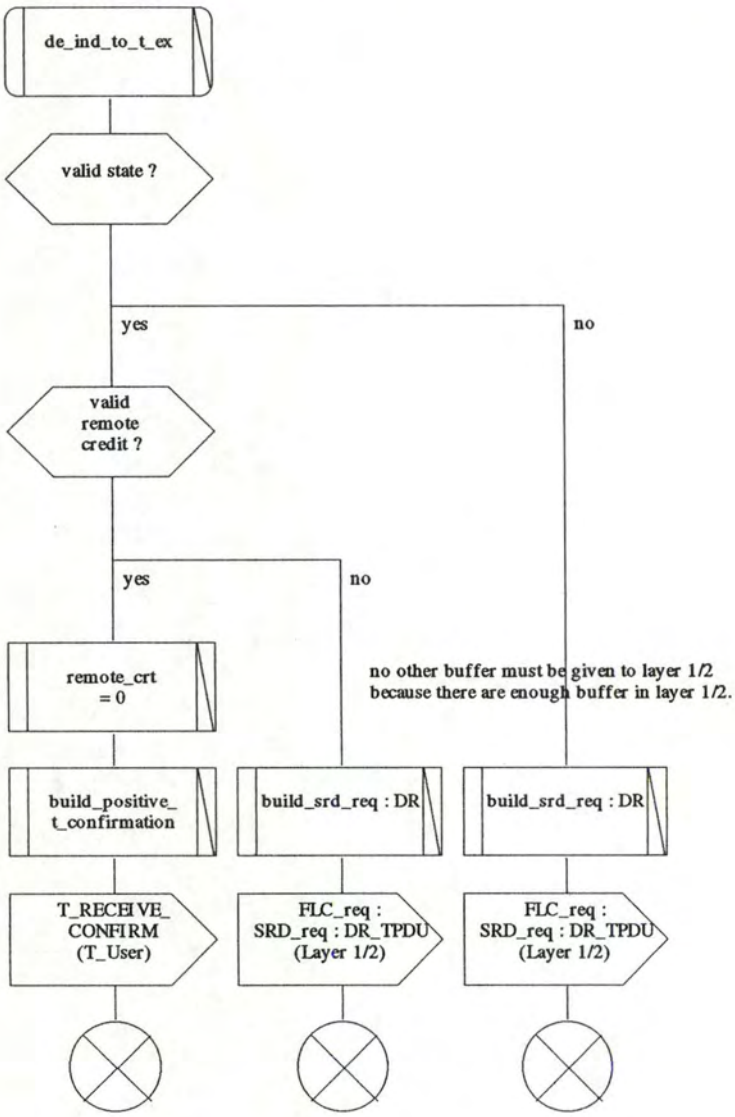

t_disconnect_ex
valid cn_id ?
yes
no
valid state ?
yes
no
update_aa_list
revise_send_
eom_fifo
all t_send_eom_services that are currently terminatetd with or without ACK and the t_receive all not sendet t_send_eom service demands, were confirmed negative. with "LOCAL ABORT".
build_srd_req : DR
build_negative_
t_confirmation
build_negative_
t_confirmation
FLC_req :
SRD_req : DR-TPDU
(Layer 1/2)
T_DISCONNECT_
CONFIRM
(T_User)
T_DISCONNECT_
CONFIRM
(T_User)

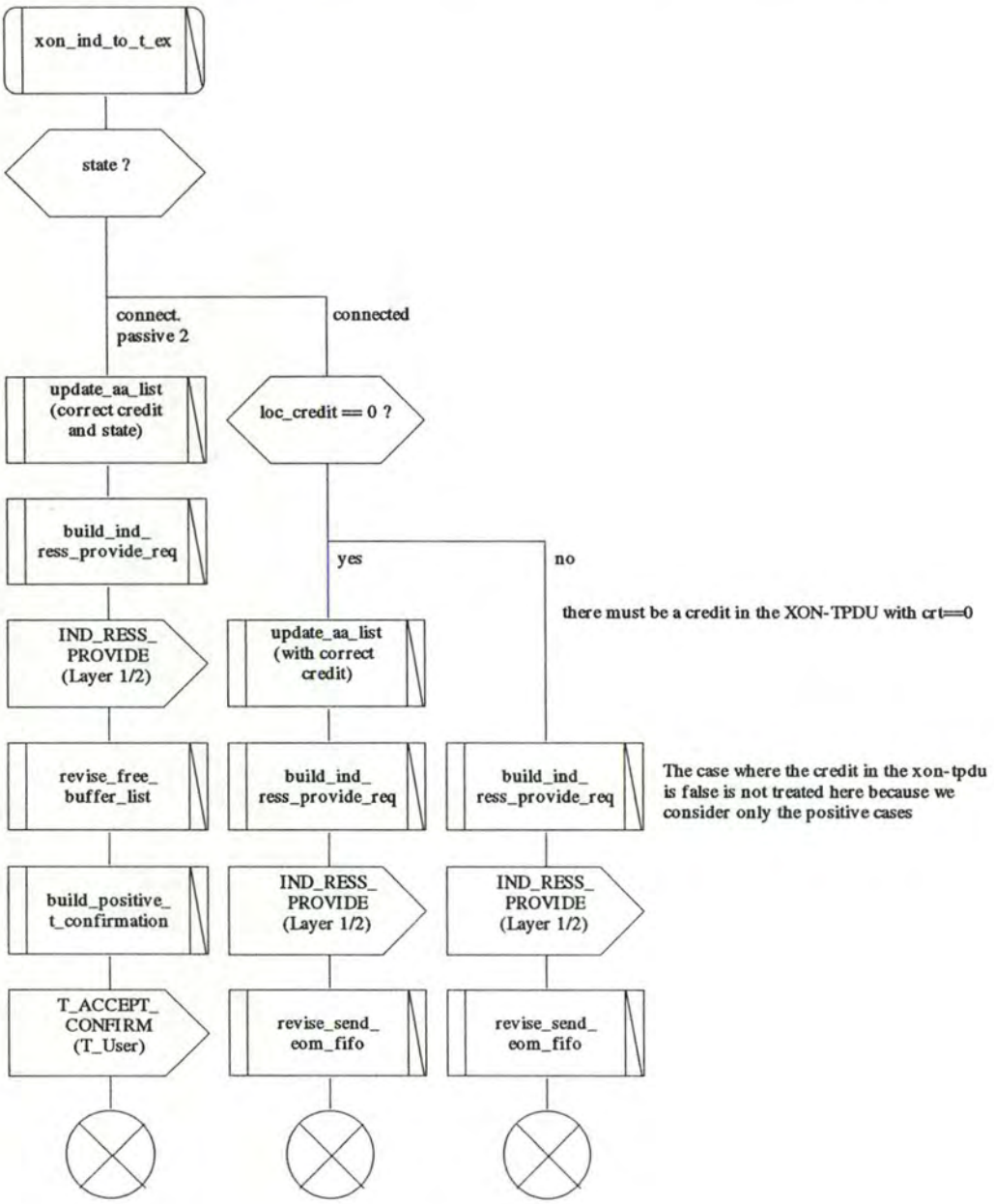

cr_ind_to_t_ex
valid state ?
yes
no
check_
connection_
parameters
return value
=
valid TSAP ?
yes
no
state_flag
=
passive
connecting1
here isn't it necessary to
prepare a buffer for DC.ind
because when DR directly
after CR in the beginning it is not
answerd by DC.ind;
build_ind._
ress._provide
FMA_req :
IND_RESS_PROVIDE
(Layer 1/2)
prepare a buffer for DC.ind
build_positive_
t_confirmation
build_srd_req : DR
build_srd_req : DR
T_AWAIT_
CONNECT_CONFIRM
(T_User)
FLC_req :
SRD_req : DR_TPDU
(Layer 1/2)
FLC_req :
SRD_req : DR_TPDU
(Layer 1/2)
The DR-TPDU is sendet over
LSAP 0 because the connection
wasn't already established.

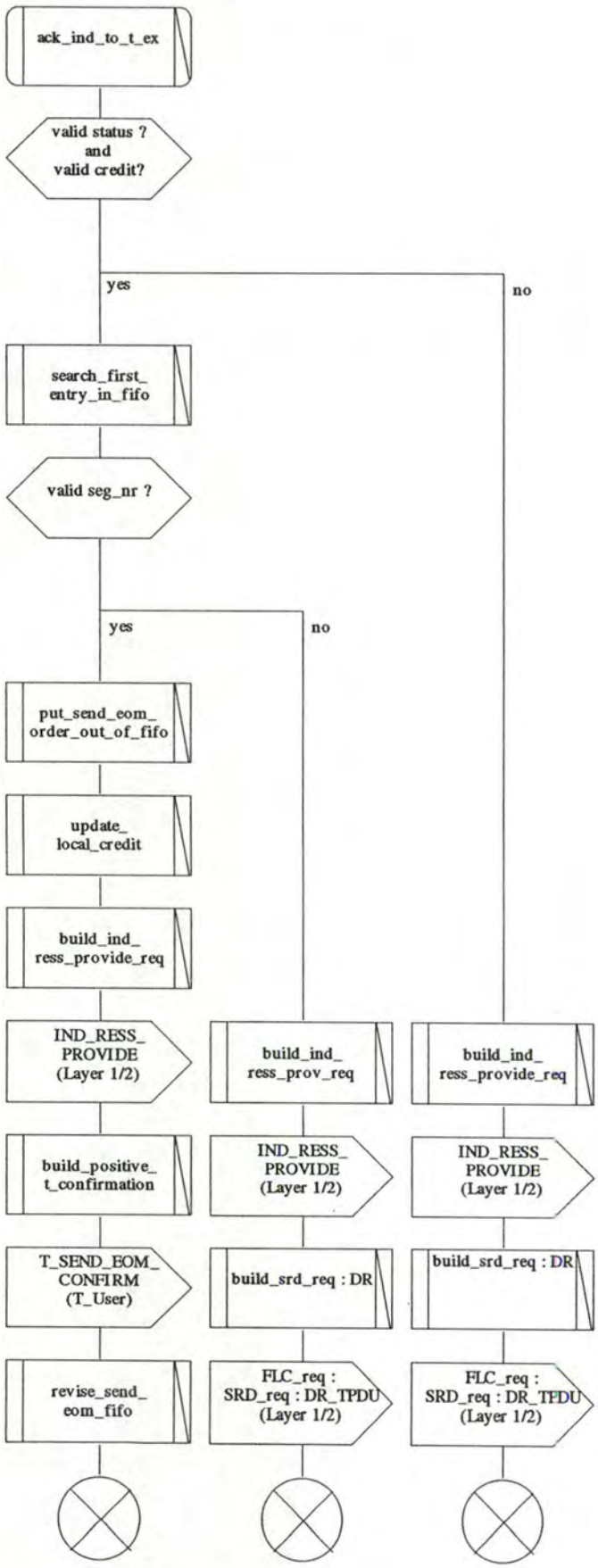


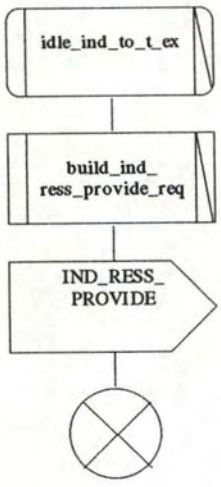
We consider only the cases, where are no errors.
So the dr_ind cames only in disconnecting phase.
when all other services (excepting t_await_disc.)
have been finished.


dc_ind_to_t_ex
We consider only the cases, where are no errors.
So the dr_ind cames only in disconnecting phase.
when all other services (excepting t_await_disc.)
have been finished.
state_flag
==
disc. active
yes
no
build_
ind_res_with_req
protocol_
error
FMA_req :
IND_RES_WITH
(Layer 1/2)

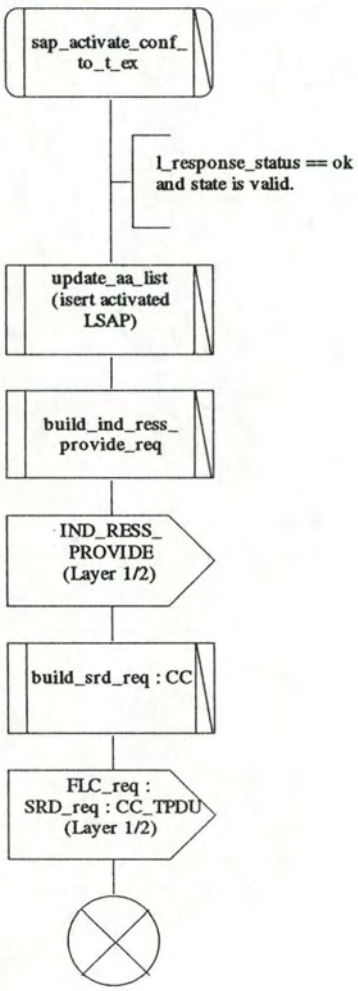

de_ind_to_t_ex
valid state ?
yes
no
valid remote credit ?
yes
no
remote_crt = 0
no other buffer must be given to layer 1/2 because there are enough buffer in layer 1/2.
build_positive_ t_confirmation
build_srd_req : DR
build_srd_req : DR
T_RECEIVE_ CONFIRM (T_User)
FLC_req : SRD_req : DR_TPDU (Layer 1/2)
FLC_req : SRD_req : DR_TPDU (Layer 1/2)

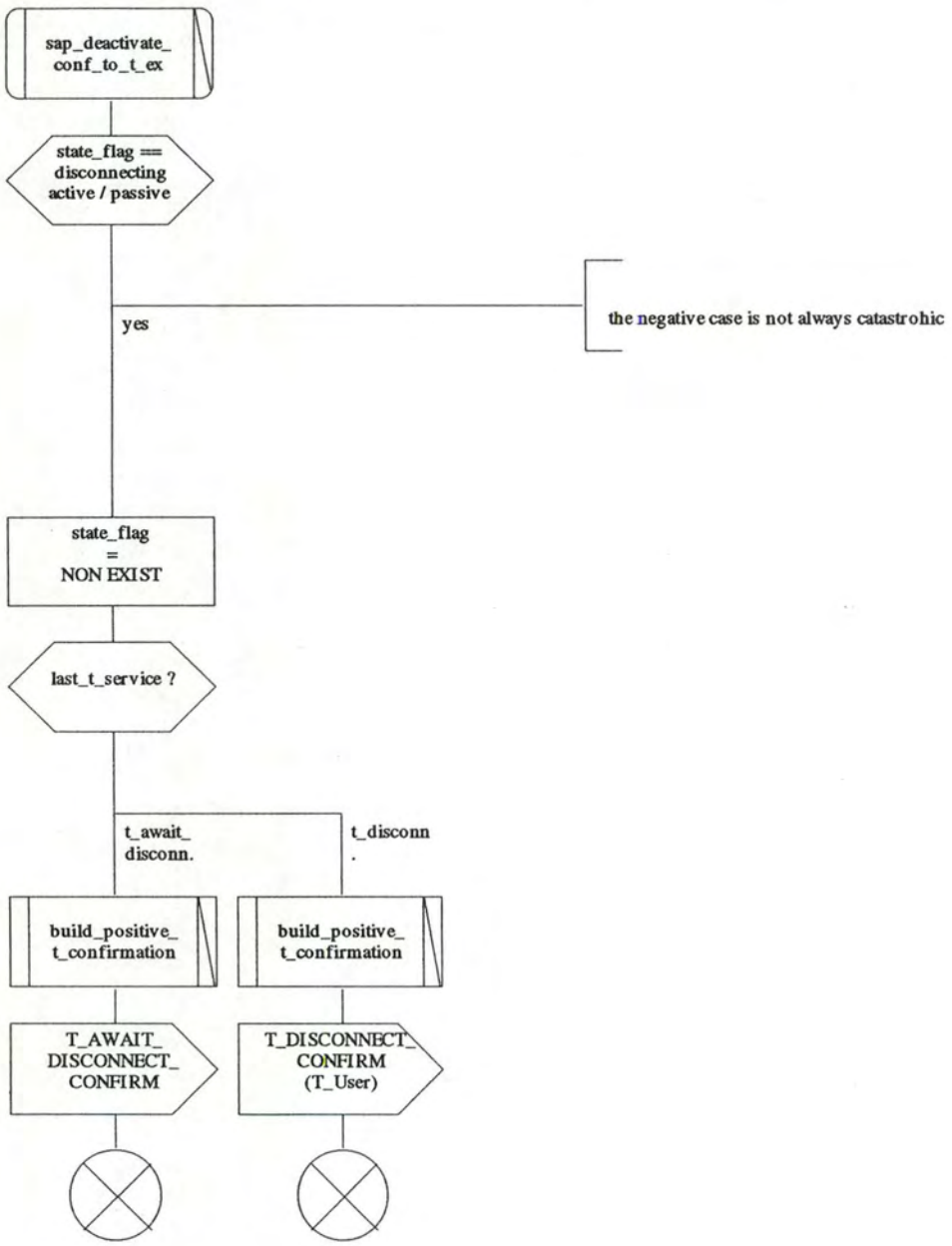

xon_ind_to_t_ex
state ?
connect.
passive 2
connected
update_aa_list
(correct credit
and state)
loc_credit == 0 ?
build_ind_
ress_provide_req
yes
no
there must be a credit in the XON-TPDU with crt==0
IND_RESS_
PROVIDE
(Layer 1/2)
update_aa_list
(with correct
credit)
revise_free_
buffer_list
build_ind_
ress_provide_req
build_ind_
ress_provide_req
The case where the credit in the xon-tpdu is false is not treated here because we consider only the positive cases
build_positive_
t_confirmation
IND_RESS_
PROVIDE
(Layer 1/2)
IND_RESS_
PROVIDE
(Layer 1/2)
T_ACCEPT_
CONFIRM
(T_User)
revise_send_
eom_fifo
revise_send_
eom_fifo

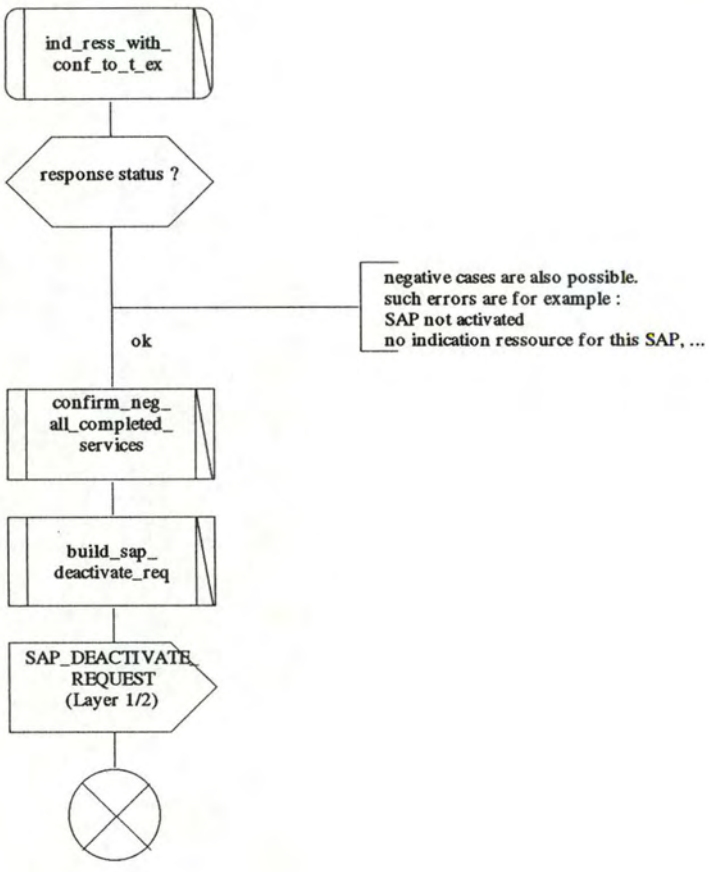

ack_ind_to_t_ex
valid status ?
and
valid credit?
yes
no
search_first_
entry_in_fifo
valid seg_nr ?
yes
no
put_send_eom_
order_out_of_fifo
update_
local_credit
build_ind_
ress_provide_req
IND_RESS_
PROVIDE
(Layer 1/2)
build_ind_
ress_prov_req
build_ind_
ress_provide_req
build_positive_
t_confirmation
IND_RESS_
PROVIDE
(Layer 1/2)
IND_RESS_
PROVIDE
(Layer 1/2)
T_SEND_EOM_
CONFIRM
(T_User)
build_srd_req : DR
build_srd_req : DR
revise_send_
eom_fifo
FLC_req :
SRD_req : DR_TPDU
(Layer 1/2)
FLC_req :
SRD_req : DR_TPDU
(Layer 1/2)

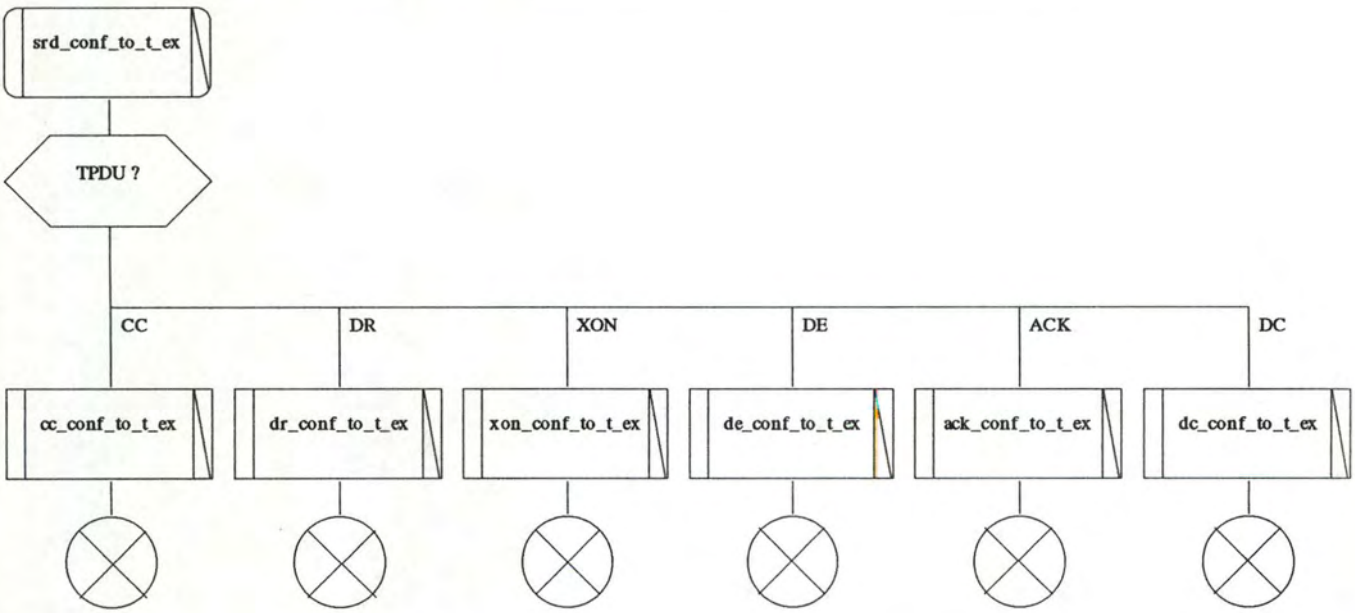


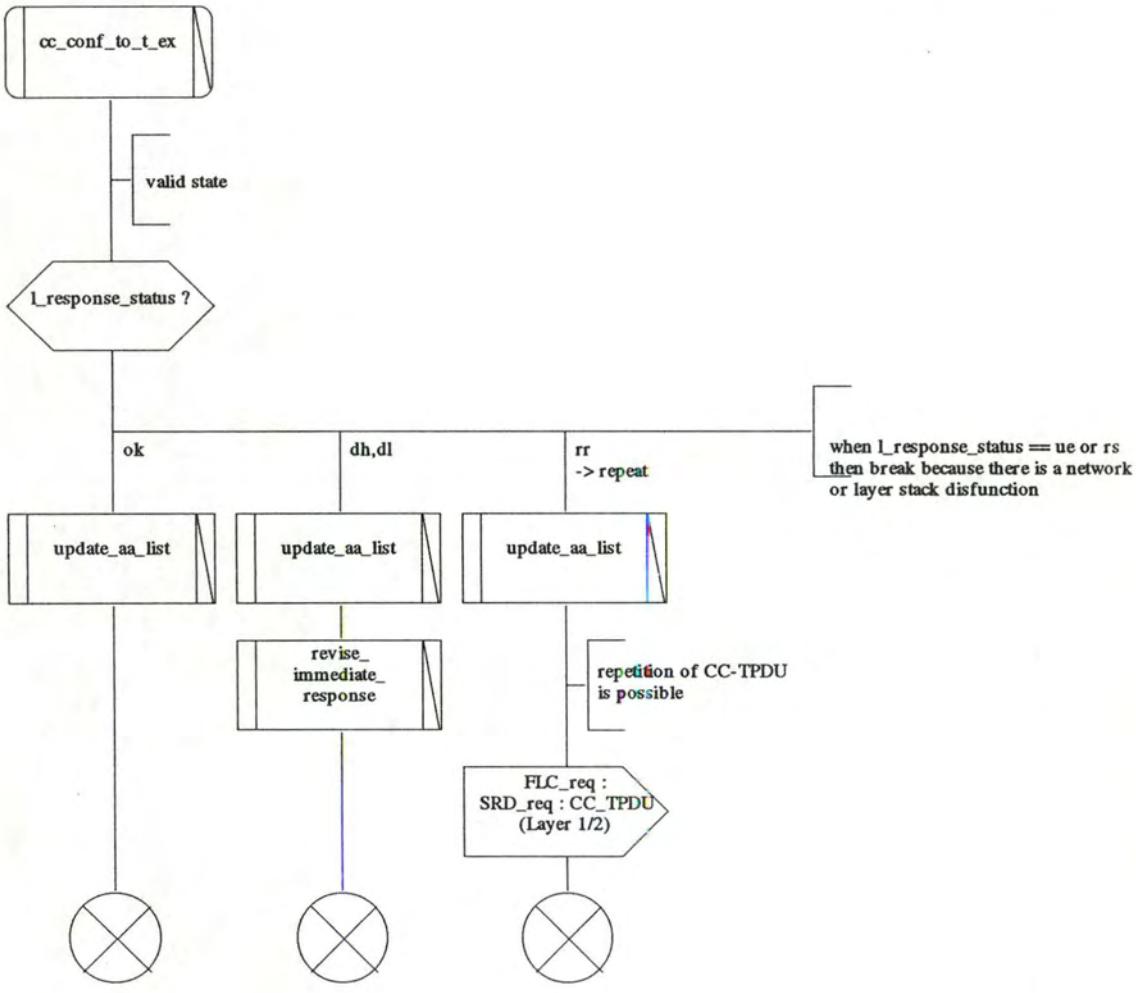
- we don't send an idle-tpdu
- we must only be capable to receive an idle tpdu
- this is only necessary when we communicate with Sinec

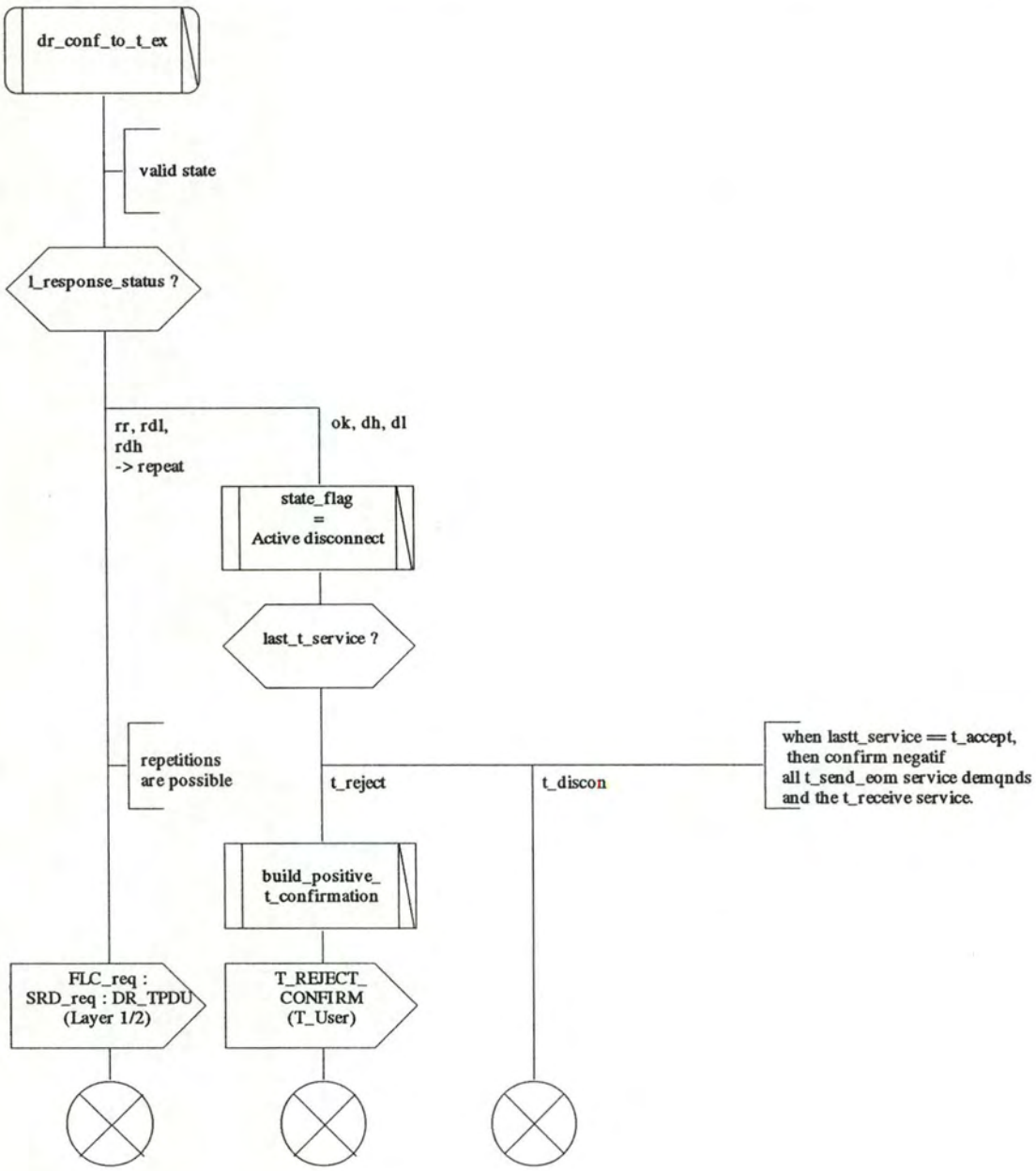

sap_activate_conf_
to_t_ex
l_response_status == ok
and state is valid.
update_aa_list
(isert activated
LSAP)
build_ind_ress_
provide_req
IND_RESS_
PROVIDE
(Layer 1/2)
build_srd_req : CC
FLC_req :
SRD_req : CC_TPDU
(Layer 1/2)

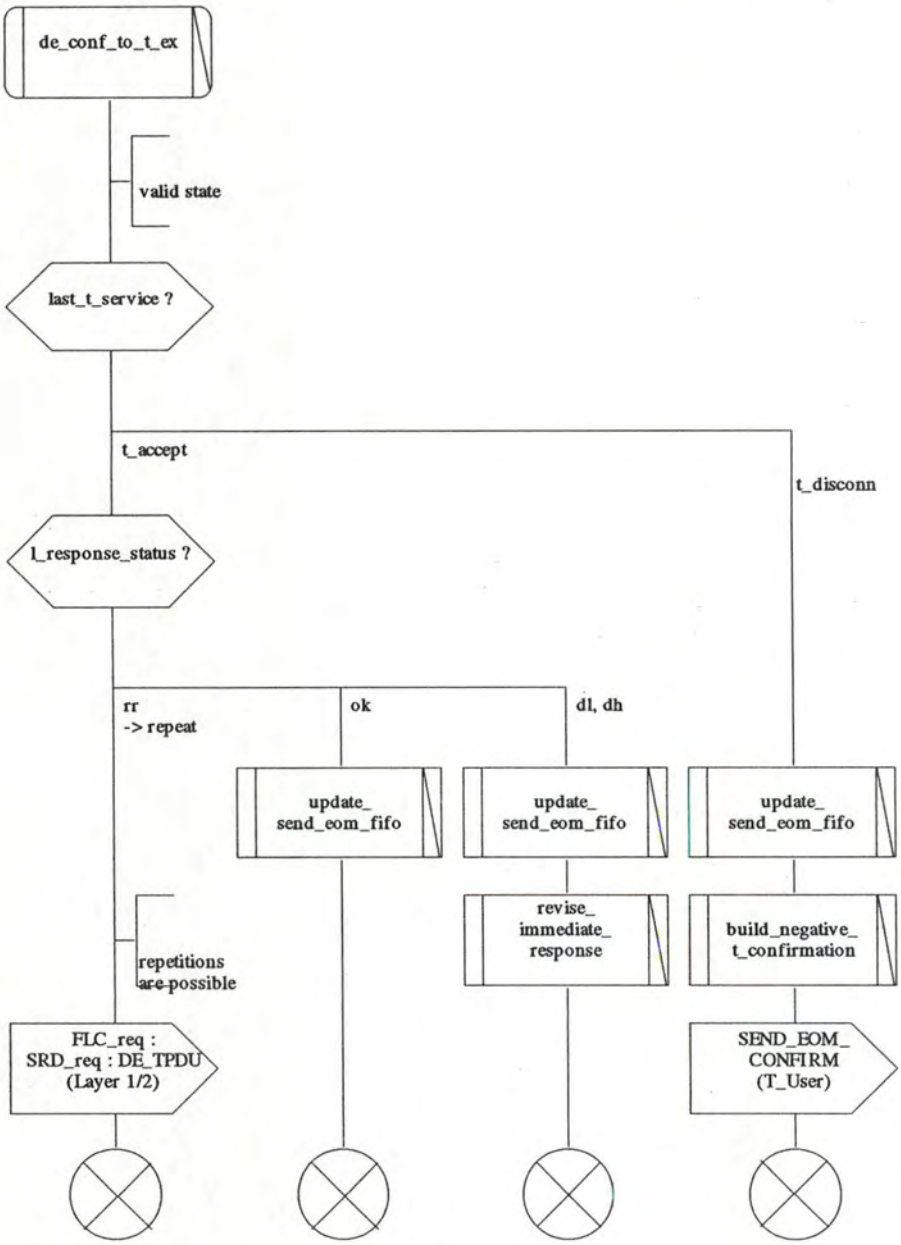

sap_deactivate_
conf_to_t_ex
state_flag ==
disconnecting
active / passive
yes
the negative case is not always catastrohic
state_flag
=
NON EXIST
last_t_service ?
t_await_
disconn.
t_disconn
.
build_positive_
t_confirmation
build_positive_
t_confirmation
T_AWAIT_
DISCONNECT_
CONFIRM
T_DISCONNECT_
CONFIRM
(T_User)

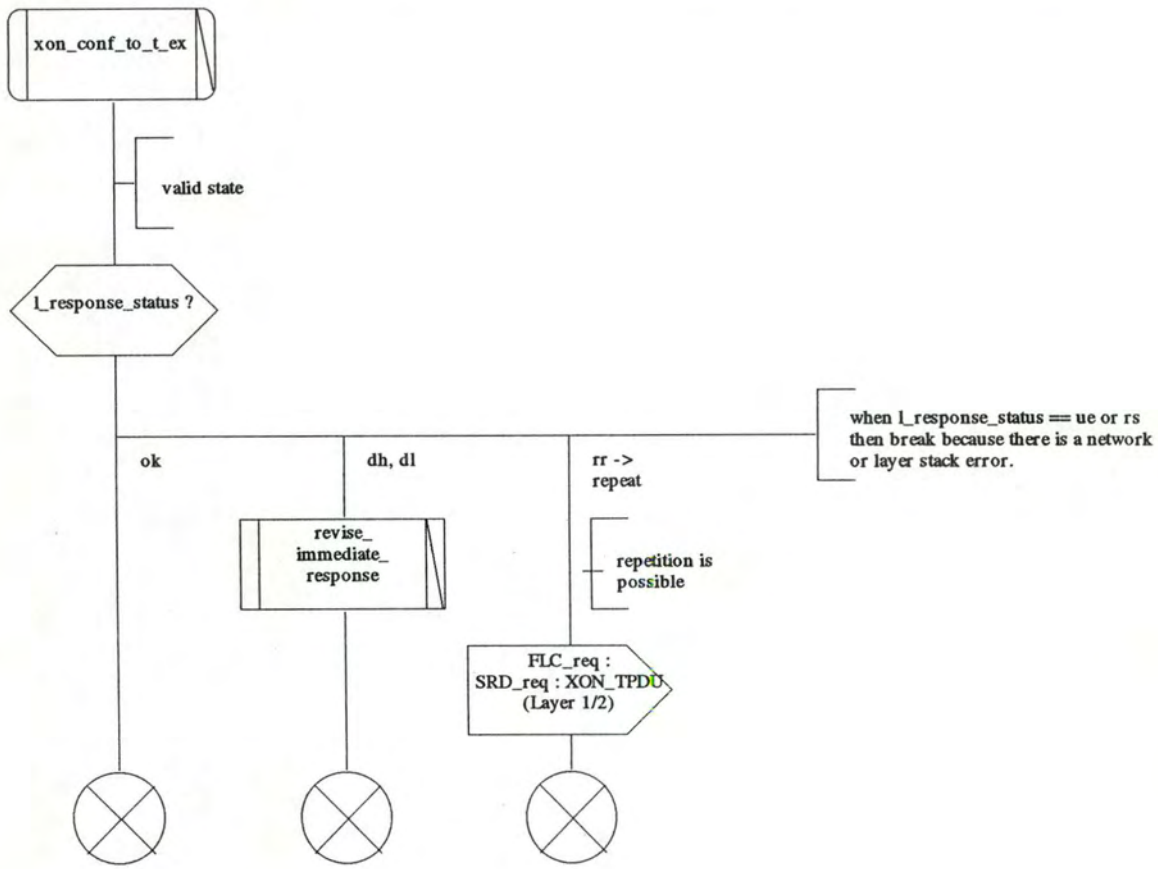

ind_ress_with_
conf_to_t_ex
response status ?
negative cases are also possible.
such errors are for example :
SAP not activated
no indication ressource for this SAP, ...
ok
confirm_neg_
all_completed_
services
build_sap_
deactivate_req
SAP_DEACTIVATE_
REQUEST
(Layer 1/2)

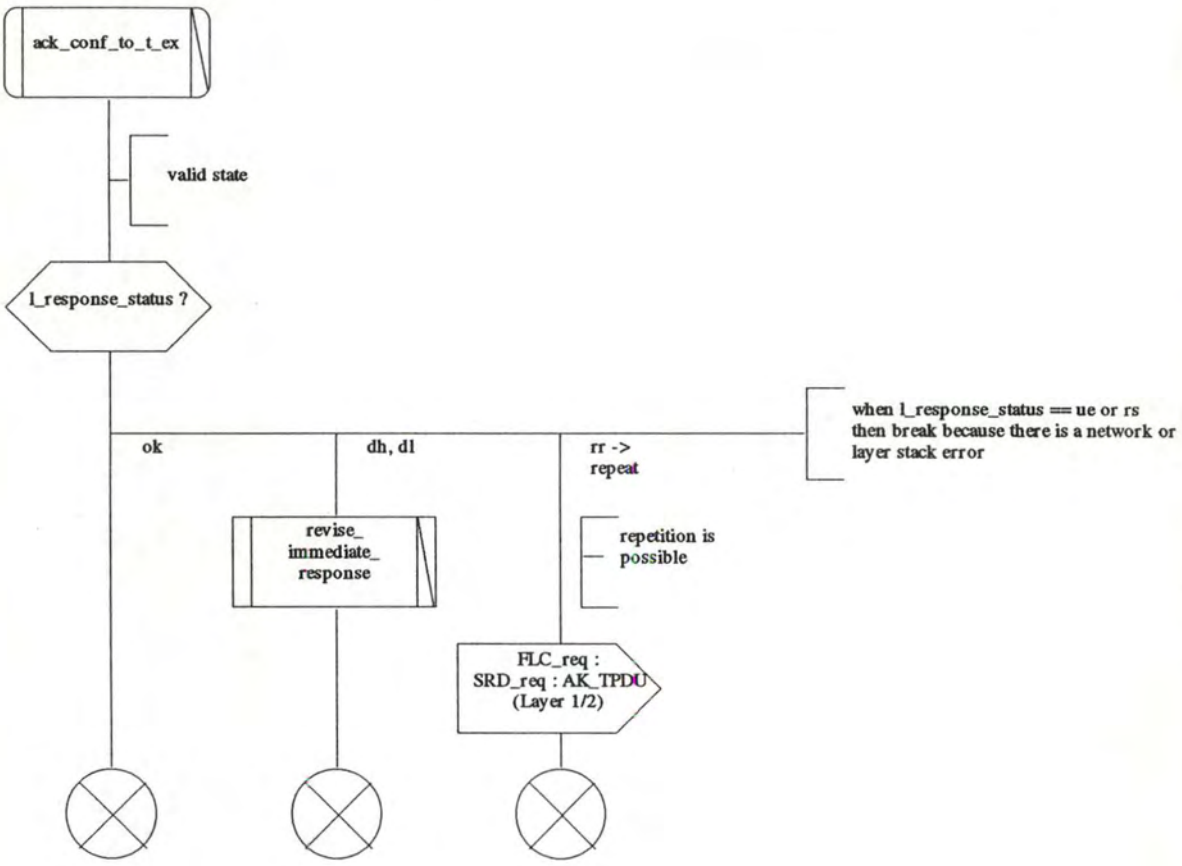

srd_conf_to_t_ex
TPDU ?
CC
DR
XON
DE
ACK
DC
cc_conf_to_t_ex
dr_conf_to_t_ex
xon_conf_to_t_ex
de_conf_to_t_ex
ack_conf_to_t_ex
dc_conf_to_t_ex

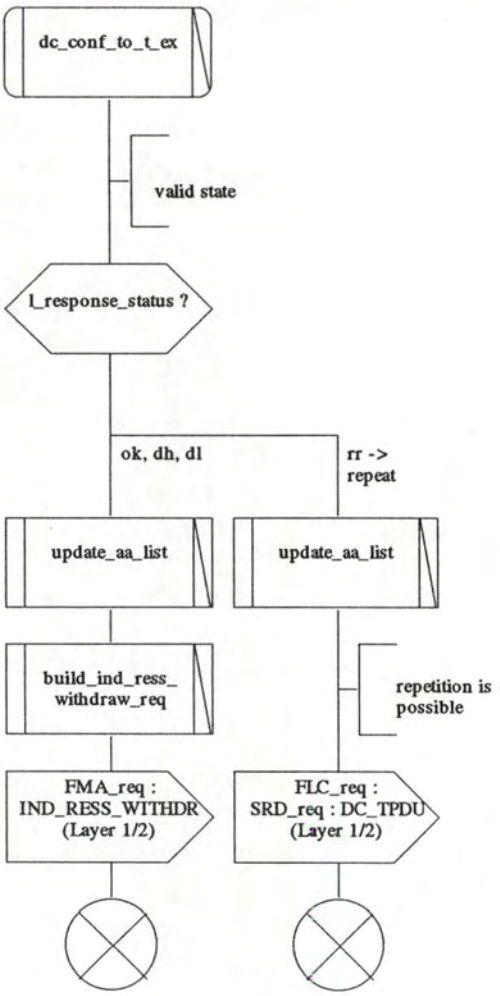

cc_conf_to_t_ex
valid state
l_response_status ?
ok
dh,dl
rr
-> repeat
when l_response_status == ue or rs
then break because there is a network
or layer stack disfunction
update_aa_list
update_aa_list
update_aa_list
revise_
immediate_
response
repetition of CC-TPDU
is possible
FLC_req :
SRD_req : CC_TPDU
(Layer 1/2)

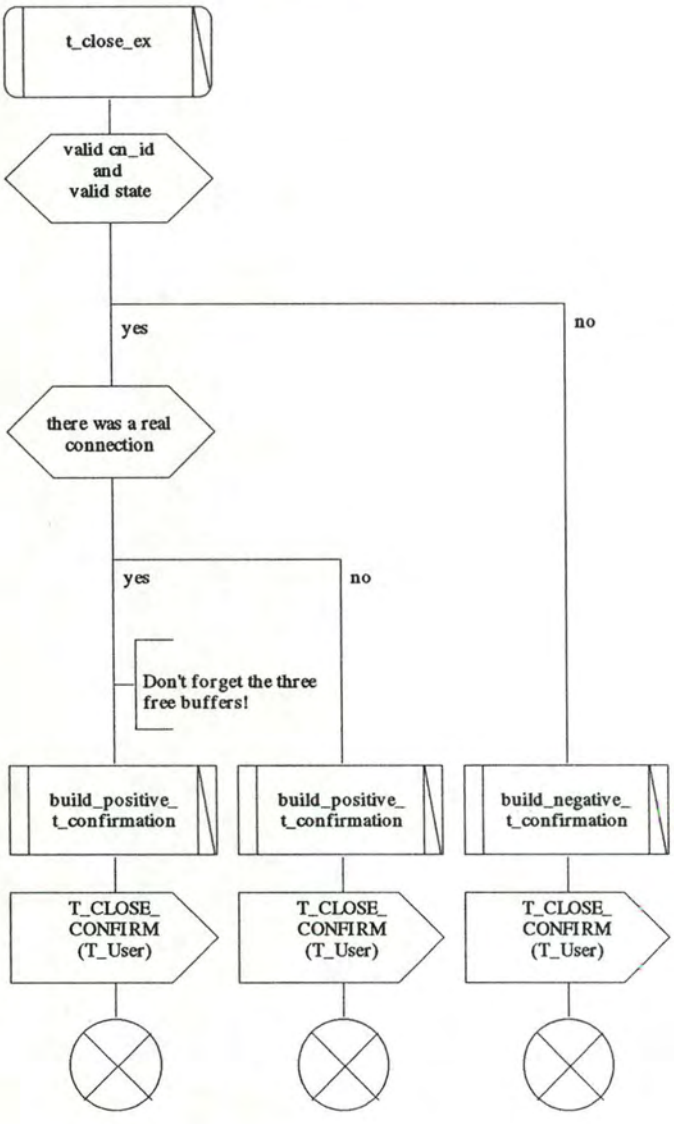

dr_conf_to_t_ex
valid state
l_response_status ?
rr, rdl,
rdh
-> repeat
ok, dh, dl
state_flag
=
Active disconnect
last_t_service ?
repetitions
are possible
t_reject
t_discon
when lastt_service == t_accept,
then confirm negatif
all t_send_eom service demqnds
and the t_receive service.
build_positive_
t_confirmation
FLC_req :
SRD_req : DR_TPDU
(Layer 1/2)
T_REJECT_
CONFIRM
(T_User)

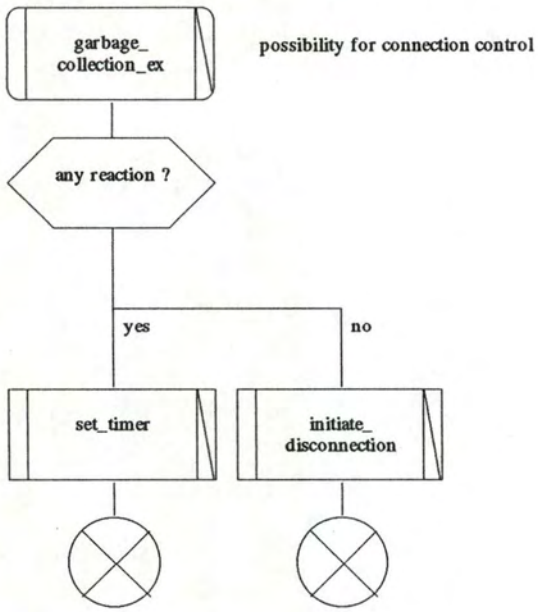

de_conf_to_t_ex
valid state
last_t_service ?
t_accept
t_disconn
l_response_status ?
rr
-> repeat
ok
dl, dh
update_
send_eom_fifo
update_
send_eom_fifo
update_
send_eom_fifo
revise_
immediate_
response
build_negative_
t_confirmation
repetitions
are possible
FLC_req :
SRD_req : DE_TPDU
(Layer 1/2)
SEND_EOM_
CONFIRM
(T_User)


xon_conf_to_t_ex
valid state
l_response_status ?
when l_response_status == ue or rs
then break because there is a network
or layer stack error.
ok
dh, dl
rr ->
repeat
revise_
immediate_
response
repetition is
possible
FLC_req :
SRD_req : XON_TPDU
(Layer 1/2)


ack_conf_to_t_ex
valid state
l_response_status ?
when l_response_status == ue or rs
then break because there is a network or
layer stack error
ok
dh, dl
rr ->
repeat
revise_
immediate_
response
repetition is
possible
FLC_req :
SRD_req : AK_TPDU
(Layer 1/2)


dc_conf_to_t_ex
valid state
l_response_status ?
ok, dh, dl
rr ->
repeat
update_aa_list
update_aa_list
build_ind_ress_
withdraw_req
repetition is
possible
FMA_req :
IND_RESS_WITHDR
(Layer 1/2)
FLC_req :
SRD_req : DC_TPDU
(Layer 1/2)


t_close_ex
valid cn_id
and
valid state
yes
no
there was a real
connection
yes
no
Don't forget the three
free buffers!
build_positive_
t_confirmation
build_positive_
t_confirmation
build_negative_
t_confirmation
T_CLOSE_
CONFIRM
(T_User)
T_CLOSE_
CONFIRM
(T_User)
T_CLOSE_
CONFIRM
(T_User)


garbage_
collection_ex
possibility for connection control
any reaction ?
yes
no
set_timer
initiate_
disconnection